Greenplum Architecture, Administration, and Implementation

Lab Guide

March 2016

Month 2011

EMC Education Services

Copyright

Copyright © 1996, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015 EMC Corporation. All Rights Reserved. EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED “AS IS.” EMC CORPORATION MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

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

EMC2, EMC, Data Domain, RSA, EMC Centera, EMC ControlCenter, EMC LifeLine, EMC OnCourse, EMC Proven, EMC Snap, EMC SourceOne, EMC Storage Administrator, Acartus, Access Logix, AdvantEdge, AlphaStor, ApplicationXtender, ArchiveXtender, Atmos, Authentica, Authentic Problems, Automated Resource Manager, AutoStart, AutoSwap, AVALONidm, Avamar, Captiva, Catalog Solution, C-Clip, Celerra, Celerra Replicator, Centera, CenterStage, CentraStar, ClaimPack, ClaimsEditor, CLARiiON, ClientPak, Codebook Correlation Technology, Common Information Model, Configuration Intelligence, Configuresoft, Connectrix, CopyCross, CopyPoint, Dantz, DatabaseXtender, Direct Matrix Architecture, DiskXtender, DiskXtender 2000, Document Sciences, Documentum, elnput, E-Lab, EmailXaminer, EmailXtender, Enginuity, eRoom, Event Explorer, FarPoint, FirstPass, FLARE, FormWare, Geosynchrony, Global File Virtualization, Graphic Visualization, Greenplum, HighRoad, HomeBase, InfoMover, Infoscape, Infra, InputAccel, InputAccel Express, Invista, Ionix, ISIS, Max Retriever, MediaStor, MirrorView, Navisphere, NetWorker, nLayers, OnAlert, OpenScale, PixTools, Powerlink, PowerPath, PowerSnap, QuickScan, Rainfinity, RepliCare, RepliStor, ResourcePak, Retrospect, RSA, the RSA logo, SafeLine, SAN Advisor, SAN Copy, SAN Manager, Smarts, SnapImage, SnapSure, SnapView, SRDF, StorageScope, SupportMate, SymmAPI, SymmEnabler, Symmetrix, Symmetrix DMX, Symmetrix VMAX, TimeFinder, UltraFlex, UltraPoint, UltraScale, Unisphere, VMAX, Vblock, Viewlets, Virtual Matrix, Virtual Matrix Architecture, Virtual Provisioning, VisualSAN, VisualSRM, Voyence, VPLEX, VSAM-Assist, WebXtender, xPression, xPresso, YottaYotta, the EMC logo, and where information lives, are registered trademarks or trademarks of EMC Corporation in the United States and other countries.

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

© Copyright 2015 EMC Corporation. All rights reserved. Published in the USA.

Revision Date: May 2015

Revision Number: MR-1CN-GRNADM.4.3.4

# Document Revision History

|  |  |  |
| --- | --- | --- |
| **Rev #** | **File Name** | **Date** |
| 1 | MR-1CN-GRNADM\_LabGuide.docx | 03/11 |
| 2 | MR-1CN-GRNADM\_v2\_LabGuide.docx | 05/11 |
| 3 | 4326\_MR-1CN-GRNADM\_LabGuide.docx | 02/12 |
| 4 | MR-1CN-GRNADM\_v2.1\_LabGuide | 04/12 |
| 5 | MR-1CN-GRNADM\_v2.2\_LabGuide | 11/13 |
| 6 | MR-1CN-GRNADM\_4.3.4\_LabGuide | 05/15 |
| 7 | MR-1CN-GRNADM\_4.3.4\_LabGuide modifications to support Piv0tal lab environment | 03/02/16 |

# Table of Contents

[Copyright 2](#_Toc418679024)

[Document Revision History 3](#_Toc418679025)

[Table of Contents 4](#_Toc418679026)

[Introduction to Lab Exercises 7](#_Toc418679027)

[Lab 1: Greenplum Product Overview 13](#_Toc418679028)

[Lab 1: Task – Greenplum Product Review 14](#_Toc418679029)

[Lab 2: Systems Preparation and Verification 17](#_Toc418679030)

[Lab 2: Task 1 – Install the Pivotal Greenplum Database Binaries 18](#_Toc418679031)

[Lab 2: Task 2 – Install and Configure Greenplum on all Hosts 27](#_Toc418679032)

[Lab 2: Task 3 – Creating the Data Storage Areas 31](#_Toc418679033)

[Lab 2: Task 4 – Synchronizing System Clocks 34](#_Toc418679034)

[Lab 2: Task 5 – Perform System Verification Tests 42](#_Toc418679035)

[Lab 3: Pivotal Greenplum Database Initialization 53](#_Toc418679036)

[Lab 3: Task 1 – Initialize the Database 54](#_Toc418679037)

[Lab 3: Task 2 – Delete the Database and Initialize a New Database with One Command 74](#_Toc418679038)

[Lab 4: Using the PSQL Client and Greenplum Utilities 83](#_Toc418679039)

[Lab 4: Task 1 – Access the Database with psql 84](#_Toc418679040)

[Lab 4: Task 2 – Install and Configure Pivotal Greenplum Command Center 91](#_Toc418679041)

[Lab 4: Task 3 – Navigating Pivotal Greenplum Command Center 107](#_Toc418679042)

[Lab 5: Greenplum Database Server Configuration 113](#_Toc418679043)

[Lab 5: Task 1 – Set Configuration Parameters 114](#_Toc418679044)

[Lab 5: Task 2 – Configure Host-Based Authentication 122](#_Toc418679045)

[Lab 6: Database Internals 125](#_Toc418679046)

[Lab 6: Task 1 – Explore Database Files on Disk 126](#_Toc418679047)

[Lab 6: Task 2 – Examine Server Processes 131](#_Toc418679048)

[Lab 7: Data Definition Language 135](#_Toc418679049)

[Lab 7: Task 1 – Create Databases and Schemas 136](#_Toc418679050)

[Lab 7: Task 2 – Create Tables 141](#_Toc418679051)

[Lab 7: Task 3 – View Indexes and Schemas 144](#_Toc418679052)

[Lab 8: Data Manipulation Language and Data Query 147](#_Toc418679053)

[Lab 8: Task 1 – Insert, Update, and Delete Records 148](#_Toc418679054)

[Lab 8: Task 2 – Accessing Data to Generate Reports 151](#_Toc418679055)

[Lab 9: Roles, Privileges, and Resources 159](#_Toc418679056)

[Lab 9: Task 1 – Manage User and Group Roles 160](#_Toc418679057)

[Lab 9: Task 2 – Manage Resource Queues 165](#_Toc418679058)

[Lab 10: Controlling Access 171](#_Toc418679059)

[Lab 10: Task – Implement Basic Security at the Group Level 172](#_Toc418679060)

[Lab 11: Table Management 179](#_Toc418679061)

[Lab 11: Task 1 – Creating Temporary Tables 180](#_Toc418679062)

[Lab 11: Task 2 – Creating Compressed Tables 184](#_Toc418679063)

[Lab 11: Task 3 – Creating a Column-Wise Storage Table 187](#_Toc418679064)

[Lab 12: Data Loading 195](#_Toc418679065)

[Lab 12: Task 1 – Create a Dimension Table and Load Data Using COPY 196](#_Toc418679066)

[Lab 12: Task 2 – Create a Dimension Table and Load Data Using External Tables 199](#_Toc418679067)

[Lab 12: Task 3 – Create a Fact Table and Load Data using gpfdist 202](#_Toc418679068)

[Lab 12: Task 4 – Create a Writeable External Table to Load Data into a Different Database using gpfdist 209](#_Toc418679069)

[Lab 12: Task 5 – Loading a Compressed File using gpfdist 216](#_Toc418679070)

[Lab 13: Table Partitioning 219](#_Toc418679071)

[Lab 13: Task – Create and Manage Table Partitions 220](#_Toc418679072)

[Lab 14: Managing the Greenplum Database 233](#_Toc418679073)

[Lab 14: Task – Perform System Administration Tasks on the Greenplum Environment 234](#_Toc418679074)

[Lab 15: Backups and Restores 249](#_Toc418679075)

[Lab 15: Task 1 – Backups and Restores 250](#_Toc418679076)

[Lab 15: Task 2 – Incremental Backup and Restore 262](#_Toc418679077)

[Lab 15: Task 3 – Recover from a Failed Master (OPTIONAL) 272](#_Toc418679078)

[Lab 16: Data Modeling 295](#_Toc418679079)

[Lab 16: Task – Create the datamart Database and Objects 296](#_Toc418679080)

[Lab 17: Physical Design Decisions 299](#_Toc418679081)

[Lab 17: Task 1 – Create the Store Dimension 301](#_Toc418679082)

[Lab 17: Task 2 – Create the Country Dimension 303](#_Toc418679083)

[Lab 17: Task 3 – Create the Customer Dimension 305](#_Toc418679084)

[Lab 17: Task 4 – Create the Transaction Fact Table 307](#_Toc418679085)

[Lab 17: Task 5 – Load Dimension and Fact Data 310](#_Toc418679086)

[Lab 18: Database Tuning 321](#_Toc418679087)

[Lab 18: Task – Tune the Database and Queries 322](#_Toc418679088)

[Lab 19: Explain the EXPLAIN Plan – Analyzing Queries 335](#_Toc418679089)

[Lab 19: Task 1– Configure Databases in pgAdmin III 336](#_Toc418679090)

[Lab 19: Task 2 – Analyze Queries 341](#_Toc418679091)

[Lab 20: Improve Performance with Statistics 349](#_Toc418679092)

[Lab 20: Task – Gather Statistics and Analyze Queries 350](#_Toc418679093)

[Lab 21: Indexing Strategies 357](#_Toc418679094)

[Lab 21: Task – Create Indexes to Support Queries 358](#_Toc418679095)

[Lab 22: Advanced Reporting Using OLAP 365](#_Toc418679096)

[Lab 22: Task – Advanced Reporting Using OLAP 366](#_Toc418679097)

[Lab 23: PostgreSQL Functions 377](#_Toc418679098)

[Lab 23: Task 1 – Create SQL Functions 378](#_Toc418679099)

[Lab 23: Task 2 – Create PL/pgSQL Functions 381](#_Toc418679100)

# Introduction to Lab Exercises

|  |  |
| --- | --- |
| **Purpose:** | The lab exercises contained in this lab guide give you the opportunity to reinforce what you have learned in the Greenplum Architecture, Administration, and Implementation course. The lab exercises are designed to run in the environment provided. |
|  | |
| **Tasks:** | Students will access the lab environment by connecting to and logging in to the Virtual Data Center (VDC) environment in Lab Manager. Students will have access to their own VDC configuration using the credentials provided by the instructor.  The virtual lab environment consists of five virtual machines with the following:   1. mdw (172.16.1.11) – Centos 6.5 x64 server with the Greenplum Database 4.3 software installed. 2. smdw (172.16.1.14) – Centos 6.5 x64 server acting as the standby server for the configuration. 3. sdw1 (172.16.1.12) – Centos 6.5 x64 server acting as the first segment server in the cluster. 4. sdw2 (172.16.1.13) – Centos 6.5 x64 server acting as the segment sever in the cluster.     Students will perform lab exercises connecting by PuTTY or other terminal application as necessary. |

Lab Introduction

**Note:** Lab access is available in the Instructor Led Training (ILT) program. Your instructor has provided you with a .zip file named somethingorother.zip. Unzip this file to the desktop on your computer.

| **Step** | **Action** |
| --- | --- |
|  | On a windows platform launch Putty. Enter 172.16.1.11 in the Host Name ( or IP address ) field and click **Open**.  Log in with the following credentials: Username: **root** Password: **Piv0tal**  If you receive a PuTTY security alert, click Yes to continue.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1b4da082.PNG  You are now logged in to mdw.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1acb4e2b.PNG |
|  | On a Mac platform open a terminal window and ssh to the master host (mdw):  ~ $ ssh root@172.16.1.11    Log in with the following credentials: Username: **root** Password: **Piv0tal**  To make your ssh commands simpler from your Mac, you might consider creating an alias for each master host ( mdw ), the standby master host ( smdw ), segment host 1 ( sdw1 ) and segment host 2 ( sdw2 ):  alias mdw=’root@172.16.1.11’  alias smdw=’root@172.16.1.14’  alias sdw1=’root@172.16.1.12’  alias sdw2=’root@172.16.1.13’  You will need to restart your shell.  You can then ssh to the master ( mdw ) for example, using the following command:  ~ $ mdw  The ~/.bashrc file on each of the hosts has been modified to include these aliases. |

This page intentionally left blank

# 172Lab Greenplum Product Overview

|  |  |
| --- | --- |
| **Purpose:** | In this lab, you review your knowledge of Greenplum Database concepts, architecture, and components. |
|  | |
| **Tasks:** | Students perform the following task: Provide answers to the review questions provided. |
|  | |
| **References:** | Module 1 – Greenplum Fundamental Concepts   * Lesson: The Basics of Data Warehousing * Lesson: Greenplum Concepts, Features, and Benefits * Lesson: Greenplum Architecture – Shared Nothing and MPP * Lesson: Greenplum Product Overview |

Lab 1: Task – Greenplum Product Review

| **Step** | **Action** |
| --- | --- |
|  | What is the master instance and what is its purpose?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Name three clients that can connect to a Greenplum database.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | What is the role of the segment instances in a Greenplum Database system?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | What is the purpose of mirroring in a Greenplum Database?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | What is the Interconnect?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Where is table data stored in a Greenplum Database system?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | How are system management tasks performed in a Greenplum Database?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | How is data distributed in a Greenplum Database?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Which redundant components can you deploy in order to have a Greenplum Database system running without a single point of failure?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

*End of Lab Exercise*

This page intentionally left blank

# Lab Systems Preparation and Verification

|  |  |
| --- | --- |
| **Purpose:** | In this lab, you verify and prepare the operating system for installation and configuration of the Greenplum Database. |
|  | |
| **Tasks:** | Students perform the following tasks:   * + Install the Greenplum Database binaries.   + Install and configure Greenplum in all hosts that comprise the Greenplum environment.   + Create the data storage areas for the Greenplum database.   + Synchronize system clocks.   + Perform system verification tasks. |
|  | |
| **References:** | Module 2 – Database Installation and Initialization   * Lesson: Systems Preparation and Verification |

Lab 2: Task 1 – Install the Pivotal Greenplum Database Binaries

| **Step** | **Action** |
| --- | --- |
|  | If not already connected, open a terminal session to mdw and log in with the following credentials:   * Login: **root** * Password: **Piv0tal** |
|  | Using the vi editor, edit the /etc/sysctl.conf file and confirm that the parameters shown in this step correspond to the values shown. If the parameters do not exist, or the values are less than the values shown, add or modify the existing line as shown to the /etc/sysctl.conf file.  You are only verifying the values of these parameters as they exist at the bottom of the file.  Note: These parameter changes are required in your production environment to ensure better performance for the Greenplum Database.  [root@mdw ~]# **vi /etc/sysctl.conf**  kernel.shmmax = 500000000  kernel.shmmni = 4096  kernel.shmall = 4000000000  kernel.sem = 250 512000 100 2048  kernel.sysrq = 1  kernel.core\_uses\_pid = 1  kernel.msgmnb = 65536  kernel.msgmax = 65536  kernel.msgmni = 2048  net.ipv4.tcp\_syncookies = 1  net.ipv4.ip\_forward = 0  net.ipv4.conf.default.accept\_source\_route = 0  net.ipv4.tcp\_tw\_recycle = 1  net.ipv4.tcp\_max\_syn\_backlog = 4096  net.ipv4.conf.all.arp\_filter = 1  net.ipv4.ip\_local\_port\_range = 1025 65535  net.core.netdev\_max\_backlog = 10000  **net.core.rmem\_max = 2097152**  **net.core.wmem\_max = 2097152**  vm.overcommit\_memory = 2  The settings above for kernel.shmmax and kernel.shmall are minimum and maximum values.  **Note:** Commands that you edit or change are marked in bold. Each UNIX command typed in a terminal session will be preceded by the prompt. Do not type that as part of the command.  If you are not familiar with the vi editor, refer to ***Appendix A*** in your student guide. |
|  | Using the sysctl command, incorporate any changes that you have made to /etc/sysctl.conftake effect immediately. This command would normally accept a file argument, but the /etc/sysctl.conf file is the default file argument:  [root@mdw ~]# **sysctl –p**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1568cf9.PNG  This step is necessary if you have made changes to the /etc/sysctl.conf file. Confirm the settings that you added in step 2 are present. |
|  | Using the Linux cat command review /etc/security/limits.conf and verify the settings are as shown.  These settings have already been made to the file. If they have not been made to the file use vi and add them.  **\*soft nofile 65536  \*hard nofile 65536  \*soft nproc 131072 \*hard nproc 131072** |
|  | Using the vi editor, edit the /etc/hosts file and add the host names and IP addresses of all machines participating in your Greenplum environment:  **172.16.1.11 mdw # master host**  **172.16.1.14 smdw # standby host**  **172.16.1.12 sdw1 # segment one**  **172.16.1.13 sdw2 # segment two**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1598e8f.PNG  Save the modified /etc/hostsfile. |
|  | Using the scp command, copy the file /etc/hosts to all machines participating in your Greenplum environment. Start with the standby host smdw**.**  [root@mdw ~]# **scp /etc/hosts smdw:/etc/hosts**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15c8fa8.PNG  **Note:** Answer **yes** to the question**Are you sure you want to continue connecting (yes/no)** and type the root password to complete the connection.  **Note:** The root password is the same for all servers participating on the Greenplum environment. |
|  | Copy the /etc/hosts file to the first segment server, sdw1, using the scp command.  [root@mdw ~]# **scp /etc/hosts sdw1:/etc/hosts**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15d31c0.PNG  **Note:** Answer **yes** to the question ***Are you sure you want to continue connecting (yes/no)*** and type the root password to complete the connection. |
|  | Copy the /etc/hosts file to the second segment server, sdw2, using the **scp** command.  [root@mdw ~]# **scp /etc/hosts sdw2:/etc/hosts**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15dc844.PNG  **Note:** Answer **yes** to the question ***Are you sure you want to continue connecting (yes/no)*** and type the root password to complete the connection. |
|  | Typically, you would download or copy the Greenplum Database installer file to the system that will be the Greenplum Master host.  In this lab environment, the installer was preloaded in the /rawdata/Binaries directory.  Change to the /rawdata/Binaries directory and list the contents of the directory.  [root@mdw ~]# **cd /rawdata/Binaries** [root@mdw ~]# **ls** |
|  | The greenplum-db-4.3.4.0-build-1-RHEL5-x86\_64.zip file is the Greenplum binary that you will install.  Unzip the greenplum-db-4.3.4.0-build-1-RHEL5-x86\_64.zip file.  [root@mdw Binaries]# **unzip \greenplum-db-4.3.4.0-build-1-RHEL5-x86\_64.zip**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1619498.PNG  **Note:** A non-quoted backslash ‘\’ is the Bash escape character. It preserves the literal value of the next character that follows, with the exception of newline. If a \newline pair appears, and the backslash itself is not quoted, the \newline is treated as a line continuation (that is, it is removed from the input stream and effectively ignored). |
|  | Launch the installer using bash. The following is an example of the command:  [root@mdw Binaries]# **/bin/bash \ greenplum-db-4.3.4.0-build-1-RHEL5-x86\_64.bin**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML16395a1.PNG |
|  | Press the **space bar** to page through and read the license agreement. You may also press **q** at any time to stop reading the license agreement. At the end, you will be prompted to accept the license agreement.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1656f7b.PNG  Type **yes** to accept the Greenplum Database License Agreement and press **ENTER**. |
|  | Accept the default installation path and press **ENTER**.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1662301.PNG |
|  | Type **yes** and press **ENTER** to accept the default install path.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML166b042.PNG |
|  | Type **yes** and press **ENTER** to create the Greenplum Database directory.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML16777f4.PNG |
|  | This optional step applies if a previous installation of Greenplum Database could be found on the system. As this does not currently apply, press **ENTER** to continue.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML168db29.PNG |
|  | The installer has installed the Greenplum Database software and created a symbolic link named greenplum-db at the same level 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.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML16964d1.PNG |

Lab 2: Task 2 – Install and Configure Greenplum on all Hosts

| **Step** | **Action** |
| --- | --- |
|  | **Task Description**  You have just installed Greenplum Database on the master server. The following tasks will copy the Greenplum database installation from the **master** host and install the binaries to the hosts you specify. These hosts include the segment servers and the standby host.  In this task, you will execute the gpseginstall command which will:   1. Create the Greenplum system user, gpadmin, on all hosts in the cluster and set the password to changeme**.** 2. Change the ownership of the Greenplum installation directory 3. Exchange ssh keys among the master, standby, and the segments 4. Create the directories for storage data on hosts in the cluster |
|  | You should login as root to perform the following tasks. You can confirm whether or not you are logged as root with the command whoami as shown below.  Source the path from the master host Greenplum database installation as shown.  [root@mdw Binaries]# **source \ /usr/local/greenplum-db/greenplum\_path.sh**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML16ef2ce.PNG  If you open a separate terminal window, you will need to source the greenplum\_path.sh file in that window to easily access the Greenplum commands. |
|  | Use the vi editor to create a file called hostfile\_exkeys***.*** This file should contain the hostnames of each host in the Greenplum environment.  Add the following host names to hostfile\_exkeys: mdw smdw sdw1 sdw2  **Note:** if you are not familiar with the vi editor, refer to the ***Linux Basic Commands*** ***Appendix A***.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML170b8ea.PNG |
|  | Execute the gpseginstallutilityand reference the file you just created, hostfile\_exkeys***.*** The -u option will create the Greenplum administrative user in all servers. You may or may not be asked for the password for each server.  [root@mdw Binaries]# **gpseginstall -f hostfile\_exkeys \ -u gpadmin -p changeme**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML171daac.PNG |
|  | The utility may prompt you to confirm the password for all of the servers on the Greenplum cluster. If prompted you must enter the password for each server in the cluster, except the master. Press **Enter** after entering the password. If you do not see the prompt for the next password required, enter the password again and press **Enter**.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML191d5bd.PNG  When gpseginstall finishes with no errors, the bottom portion of your screen should be similar to the screen below.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML173ee7a.PNG |
|  | Verify that all hosts in the cluster are accessible and have their own copy of the Greenplum software installed. Use the gpssh command to accomplish this task.  [root@mdw Binaries]# **gpssh -f hostfile\_exkeys -e ls –F $GPHOME**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5601d99.PNG |

Lab 2: Task 3 – Creating the Data Storage Areas

| **Step** | **Action** |
| --- | --- |
|  | **Task Description**  Every Greenplum Database master and segment instance has a designated storage area on disk that is called the data directorylocation*.* This is the file system location where the directories that store segment instance data will be created. The master host needs a data storage location for the master data directory. Each segment host needs a data directory storage location for its primary segments, and another for its mirror segments. |
|  | On your master server host, change to /data directory and create the directory that will be your master data storage area as shown.  [root@mdw Binaries]# **cd /data**  [root@mdw data]# **mkdir master**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML19d5ad4.PNG |
|  | Change the ownership and the group of the new directory.  [root@mdw data]# **chown gpadmin:gpadmin /data/master**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML19e1c9c.PNG |
|  | Using gpssh, create the master data directory location on your standby master as well.  [root@mdw data]# **gpssh -h smdw -e 'mkdir /data/master'**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML19f109b.PNG |
|  | Change the ownership of the data directory togpadmin**.**  [root@mdw data]# **gpssh -h smdw -e 'chown gpadmin:gpadmin \ /data/master'**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML19fac9b.PNG |
|  | Change to the /rawdata/Binaries directory.  [root@mdw data]# **cd /rawdata/Binaries**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a6eee8.PNG |
|  | Use the vi editor to create a file called hostfile\_gpssh\_segonly***.*** This file should contain the hostnames of each segment host in the Greenplum environment.  Add the following segment host names to hostfile\_gpssh\_segonly: sdw1 sdw2  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a7ada5.PNG |
|  | Using gpssh, create the primary and mirror data directory locations on all segment hosts at once using the hostfile\_gpssh\_segonly file you just created.  [root@mdw Binaries]# **gpssh -f hostfile\_gpssh\_segonly \ -e 'mkdir /data/primary; chown gpadmin:gpadmin /data/primary'**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a933e4.PNG |
|  | Using gpssh, change the permissions on the directory /loaddata in the sdw1 server. This directory will be used in future labs.  [root@mdw Binaries]# **gpssh -h sdw1 -e 'chown -R gpadmin:gpadmin /loaddata'**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1b4f723.PNG |

Lab 2: Task 4 – Synchronizing System Clocks

| **Step** | **Action** |
| --- | --- |
|  | Greenplum recommends using NTP (Network Time Protocol) to synchronize the system clocks on all hosts that comprise your Greenplum Database system.  NTP on the segment hosts should be configured to use the master host as the primary time source and the standby master as the secondary time source. On the master and standby master hosts, configure NTP to point to your preferred time server.  On the master host, edit the /etc/ntp.conf file and perform the following tasks:  Comment the following lines  **# server 0.centos.pool.ntp.org iburst**  **# server 1.centos.pool.ntp.org iburst**  **# server 2.centos.pool.ntp.org iburst**  **# server 3.centos.pool.ntp.org iburst**  Add the line server IP address indicated as shown:  **server 172.16.1.11** |
|  | On the master host ( mdw ), modify the ntp.conf file as follows.    Save your changes and exit. |
|  | Connect to segment 1 host ( sdw1) using ssh from the current terminal window. Use vi to edit the /etc/ntp.conf file, and modify it as shown below.    Save your changes and exit vi. |
|  | Using gpscp, copy the /etc/ntp.conf file to segment 2, sdw2.  [root@sdw1 ~]# **source /usr/local/greenplum-db/greenplum\_path.sh**  [root@sdw1 ~]# **gpscp -h sdw2 /etc/ntp.conf =:/etc/ntp.conf**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1c8809d.PNG  Note that there is a space between “ntp.conf” and “=:” |
|  | Connect to the standby master server ( smdw ) using ssh. Edit /etc/ntp.conf using vi, and modify the file as shown on the picture below.    Save your changes and exit vi. |
|  | Exit from the standby master server.  [root@smdw ~]# **exit**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5557f29.PNG |
|  | Exit from the sdw1 segment server.  [root@sdw1 ~]# **exit**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML555e5a8.PNG |
|  | Synchronize the system clocks on all Greenplum hosts. This effectively starts or resets the ntpd service on all hosts within the cluster.  [root@mdw Binaries]# **gpssh -f hostfile\_exkeys -v -e 'ntpd'**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5668429.PNG |
|  | Verify the ntpd service is executing on all hosts in the cluster.  [root@mdw Binaries]# **gpssh -f hostfile\_exkeys -e 'pgrep ntp'**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5781e55.PNG |
|  | Go back to the root directory on the mdw server.  [root@mdw ~]# **cd**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML578a649.PNG |

Lab 2: Task 5 – Perform System Verification Tests

| **Step** | **Action** |
| --- | --- |
|  | You will execute several Greenplum Database utilities to verify the operating system settings and hardware performance of the servers within the Greenplum Cluster. These tests should be executed prior to initializing your Greenplum Database system.  If not already connected to the master server, open a new terminal session to mdw and log in as root with the password, Piv0tal. |
|  | Execute the gpcheck command to verify the operating system settings. This command should normally be executed against all hosts in the cluster, but due to the amount of information it provides, we will examine it first for the master server.  [root@mdw ~]# **gpcheck -h mdw**    What is the result of the command?  Are there any configuration issues that need to be fixed?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | There are likely several errors visible for the master server, mdw. The following highlights some of the I/O scheduler and the read ahead errors seen in this environment. |
|  | Update I/O scheduler value for sr0deviceusing the command below.  [root@mdw ~]# **echo deadline > /sys/block/sr0/queue/scheduler** |
|  | Another error identified by running gpcheck -h mdwis highlighted in this window. |
|  | Update readahead value for sda1 device using the command below.  [root@mdw ~]# **/sbin/blockdev --setra 16384 /dev/sdb1** |
|  | Re-execute the gpcheck command to confirm that the errors for the devices sr0 and sdb1 on mdw are no longer present. However some errors still exist on mdw. In addition all of the errors that were present on mdw are also present on smdw, sdw1 and sdw2 and need to be cleared.  [root@mdw ~]# **gpcheck -h mdw** |
|  | Execute the script, update\_block\_devices\_IO\_scheduler.sh, to update the devices on all hosts.  [root@mdw ~]# **/rawdata/solutions/lab2/update\_block\_devices\_IO\_scheduler.sh**    **Note:** This script is provided within the training environment to quickly update the master, standby, and segment servers. This is not provided by default with the Greenplum Database installation. The script uses gpssh to update the block devices on each system. You must be root to successfully execute the commands within the script. |
|  | Execute the gpcheck command to verify the configuration in all hosts. Use the /rawdata/Binaries/hostfile\_exkeys file to provide the list of all hosts in the cluster.  [root@mdw ~]# **gpcheck -f /rawdata/Binaries/hostfile\_exkeys –m mdw \**  **-s smdw** |
|  | Verify the baseline hardware performance of the segment host systems. The test you are performing will take a few minutes as it performs a stress test of memory, I/O, and network performance.  Using gpssh, execute the three commands below for all Greenplum hosts. These commands will disable the firewall among the Greenplum servers. The commands are:   1. **service iptables save** 2. **service iptables stop** 3. **chkconfig iptables off**   [root@mdw ~]# **cd /rawdata/Binaries** [root@mdw Binaries]# **gpssh –v -f hostfile\_exkeys -e 'service iptables save'**  [root@mdw Binaries]# **gpssh –v -f hostfile\_exkeys -e 'service iptables stop'**  [root@mdw Binaries]# **gpssh –v -f hostfile\_exkeys -e 'chkconfig iptables off'** |
|  | **Note:** It is necessary to either disable the firewall or to allow specific ports through the Linux firewall for specific Greenplum services. When testing the servers, the network performance test uses port 23000 by default. If this port is not open on all systems, the test will fail. Greenplum utilities may also occasionally use remote copy (rcp) services to copy files from one system to another. |
|  | Execute the gpcheckperf utility to verify the configuration. This is executed against the segment servers within the cluster. As there are only two segment hosts, we will simply specify the hostnames on the command line. If you do create a file, use the -f option followed by the filename to specify the segment servers to include in the validation tests.  Either of the following commands will work:  [root@mdw Binaries]# **gpcheckperf -h sdw1 -h sdw2 -d /data –D**  **or**  [root@mdw Binaries]# **gpcheckperf -f hostfile\_gpssh\_only -d /data –D** |
|  | C:\Users\cantot\AppData\Local\Temp\SNAGHTML5a39a1a.PNG |
|  | If you wish to perform a network only test, execute the gpcheckperf command with the -r n option. You will also need to specify a segment directory using the -d option.  What is the total bandwidth for:   * Disk Writes? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ * Disk Reads? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ * Sustainable Memory? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ * Network? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   **Note:** Network performance in the virtual environment may be slightly degraded and so you may see messages that the connection between systems does not meet the guidelines established by Greenplum as adequate for a production environment. |
|  | **Tips and Best Practices**  In a single-host environment, or virtual environment, performance will be lower than on a distributed cluster. On production systems, you should run gpcheckperf when the system is idle to get accurate performance metrics.  **Summary**  As with any database system, the performance of the Greenplum Database is dependent upon the hardware and IT infrastructure on which it is running. The Greenplum Database is comprised of several servers (or hosts) acting together as one cohesive system. The Greenplum Database’s runtime performance will be as fast as the slowest segment host in the array. It is important to know your systems’ expected level of performance before setting database performance expectations.  The Greenplum Database requires that the operating systems of the hosts on which it runs be properly tuned. These tuning parameters are especially important on large systems with complex query workloads, as queries can fail when they do not get the resources they need from the operating system. The gpcheck utility checks the OS environment of each host to ensure that they have the Greenplum recommended settings.  The expected results of the gpcheckperf tests depend on the total capacity of the server hardware you are using. If the expected disk I/O of a system is 2 GBytes per second, multiply the expected rate by the number of segment servers for the total bandwidth. If there are two segment servers, as in this environment, the expected total bandwidth is 4 GB/s.  When looking at the output from gpcheckperf, you want to make sure that your disk I/O rate is what you would expect from your hardware platform and that the memory and network bandwidth are not bottlenecks to optimal performance. (They should be comparable to or greater than disk I/O.) |

*End of Lab Exercise*

# Lab Pivotal Greenplum Database Initialization

|  |  |
| --- | --- |
| **Purpose:** | You will perform the following installation and setup tasks necessary for the Greenplum Database software to run:   * Verify that the Greenplum Database software binaries are installed on all hosts. * Ensure the required file systems are created on all hosts. * Ensure the Greenplum administrator user account is created on all hosts. * Verify that SSH keys are exchanged between all hosts. * Set environment variables on the master as needed.   Many of these tasks are now automated, but you should continue to verify that the tasks have been completed. These installation tasks should be performed from your assigned master host.  You will initialize a Greenplum Database array by examining the initialization configuration file and executing the gpinitsystem utility. You will also troubleshoot errors that may occur during initialization of a Greenplum Database array. |
|  | |
| **Tasks:** | Students perform the following tasks:   1. Initialize the database with no mirrors and standby master using the gpinitsystem utility 2. Add mirrors with the gpaddmirrors utility 3. Add a standby master with the gpinitstandby utility 4. Delete the Greenplum database using the gpdeletesystem utility 5. Initialize a new Greenplum Database with mirrors and a standby master using the gpinitsystem utility. |
|  | |
| **References:** | Module 2 – Database Installation and Initialization   * Lesson: Greenplum Database Initialization |

Lab 3: Task 1 – Initialize the Database

| **Step** | **Action** |
| --- | --- |
|  | If not already connected to mdw, open a terminal session to mdw and log in as root with the password, Piv0tal. Switch to gpadmin user.  [root@mdw ~]# **su - gpadmin**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6aebc9d.PNG |
|  | Create a directory named gpconfigs in the home directory of the gpadmin user as shown**.**  [gpadmin@mdw ~]$ **mkdir gpconfigs**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6af560e.PNG |
|  | Copy the file  /usr/local/greenplum-db/docs/cli\_help/gpconfigs/gpinitsystem\_configinto the directory /home/gpadmin/gpconfigs.  [gpadmin@mdw ~]$ **cp \ /usr/local/greenplum-db/docs/cli\_help/gpconfigs/gpinitsystem\_config ~/gpconfigs**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6afc64d.PNG |
|  | Change to the gpconfigs directory.  [gpadmin@mdw ~]$ **cd gpconfigs**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6b021f4.PNG |
|  | Create a copy of the segment only file you created earlier. This is simply for ease of use***.*** This file should contain the hostname of all segment servers participating in the Greenplum environment. This file cannot contain any extra lines or spaces.  [gpadmin@mdw ~]$ **cp /rawdata/Binaries/hostfile\_gpssh\_segonly \**  **hostfile\_gpinitsystem**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6b3b2cd.PNG |
|  | Using vi, edit the gpinitsystem\_config file. Locate the ‘**Required Parameters**’ section and replace the highlighted section with the following:  **declare -a DATA\_DIRECTORY=(/data/primary)**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6b4e65b.PNG |
|  | The final file appears as follows:  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6b58b51.PNG  This is a read-only file. Use :**wq!**to save the file.  Verify there are no comment symbols (#) in front of the line you are adding or replacing. If so, remove the comment symbol. Failure to do so may result in the initialization failing. |
|  | Run the gpinitsystem utility to create a Greenplum Database system using the values defined in gpinitsystem\_config.j  [gpadmin@mdw gpconfigs]$ **source \ /usr/local/greenplum-db/greenplum\_path.sh**  [gpadmin@mdw gpconfigs]$ **gpinitsystem -c gpinitsystem\_config \ -h hostfile\_gpinitsystem**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5c7dd9f.PNG  The utility will ask your permission to overwrite the configuration file.  Confirm by pressing **y** and press **ENTER.** |
|  | The utility displays the master, and primary segments configuration. It also asks you whether or not you want to continue with the Greenplum creation.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5c8c55f.PNG  Press **y** followed by **Enter** to continue. |
|  | This screen shows that the gpinitsystem ended with no errors. This will take a few minutes.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5cb111e.PNG |
|  | Perform the following steps on the master server (mdw):   1. Change to the gpadmin home directory, /home/gpadmin, and using the vi editor, edit the file /home/gpadmin/.bash\_profile.   [gpadmin@mdw gpconfigs] $ **cd** C:\Users\cantot\AppData\Local\Temp\SNAGHTML5ccba48.PNG   1. Add the following entries to the bottom of the file using vi:   **MASTER\_DATA\_DIRECTORY=/data/master/gpseg-1**  **export MASTER\_DATA\_DIRECTORY**  **source /usr/local/greenplum-db/greenplum\_path.sh**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5d02f96.PNG   1. Run the command below to make the changes active immediately. [gpadmin@mdw ~]$ **source .bash\_profile**   C:\Users\cantot\AppData\Local\Temp\SNAGHTML5d0b6fd.PNG |
|  | Execute the gpstateutility to verify the Greenplum instance status summary. |
|  | C:\Users\cantot\AppData\Local\Temp\SNAGHTML5d24a56.PNG |
|  | The next step is to manually add a standby master to the Greenplum Database that you just created. Run the gpinitstandby utility to do this.  [gpadmin@mdw ]$ **gpinitstandby -s smdw**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5d7c159.PNG  Press **y** to continue. |
|  | This screen shows that the gpinitstandby ended with no errors.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5ddb336.PNG |
|  | Run the gpstate -f ( the -f option displays the details of the standby master ( smdw ) ) utility to verify that the standby master was created and is synchronized with the master.  [gpadmin@mdw ]$ **gpstate -f**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5de6a16.PNG |
|  | Copy the file /home/gpadmin/.bash\_profile to smdw using the scp or gpscp command:  [gpadmin@mdw ~]$ **scp /home/gpadmin/.bash\_profile smdw:** |
|  | Create the /data/mirror directory on all segment hosts within the cluster. After creating the directory, you must change the ownership of the directory so that it is owned by the gpadmin user account.   * 1. Switch to the root user with the following command:  [gpadmin@mdw ~]$ **su -**   2. Enter the password, Piv0tal.   3. Source the greenplum\_path.sh file to access the Greenplum binaries:  [root@mdw ~]# **source \ /usr/local/greenplum-db/greenplum\_path.sh**   4. Use gpssh to connect to sdw1 and sdw2 and create the mirror directory, /data/mirror:  [root@mdw ~]# **gpssh -h sdw1 -h sdw2 -e 'mkdir /data/mirror'**   5. Change the ownership of these directories to gpadmin using the gpssh command:  [root@mdw ~]# **gpssh -h sdw1 -h sdw2 -e 'chown \ gpadmin:gpadmin /data/mirror'** Note: Steps 4 and 5 can be combined into a single command by using a semicolon to separate the mkdir and chown statements.   6. Exit from the root account session:  [root@mdw ~]# **exit**   C:\Users\cantot\AppData\Local\Temp\SNAGHTML5e2643d.PNG |
|  | The next step is to manually add mirror segments to the Greenplum Database that you just created. Run the gpaddmirrors utility to do this.  [gpadmin@mdw ]$ **gpaddmirrors**  When prompted, type in the mirror segment data directory and press **Enter**:  **/data/mirror**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5e304de.PNG |
|  | The following screen shows the configuration output from the gpaddmirrors utility:  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5e65e25.PNG |
|  | C:\Users\cantot\AppData\Local\Temp\SNAGHTML5e7076e.PNG  Press **y** to continue. |
|  | This screen shows that the gpaddmirrors ended with no errors.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5e80990.PNG |
|  | You can use the gpstate -s utility to check that the mirror segments are being synchronized with their primary segments. This utility will also indicate when the mirrors are fully synchronized.  [gpadmin@mdw ]$ **gpstate -s** |
|  | Run the gpstateutility to verify that the Greenplum instance status summary includes the new mirrors that were just added.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5ea201c.PNG |
|  | C:\Users\cantot\AppData\Local\Temp\SNAGHTML5eae77f.PNG |

Lab 3: Task 2 – Delete the Database and Initialize a New Database with One Command

| **Step** | **Action** |
| --- | --- |
|  | If not already connected to mdw, open a terminal session to mdw and log in as root with the password, Piv0tal.  Switch to gpadmin user. If you were already in the gpadmin session from the previous lab task, you can skip this step.  [root@mdw ~]# **su - gpadmin**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5ed4873.PNG |
|  | You will now delete the Greenplum Database that you created using the gpdeletesystem utility.  The -d option is pointing at the directory where the master database instance was created.  [gpadmin@mdw ~]$ **gpdeletesystem -d /data/master/gpseg-1**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5ef1d10.PNG  Press **y** to continue. |
|  | C:\Users\cantot\AppData\Local\Temp\SNAGHTML6bc1ed8.PNG  The utility will warn you one more time about deleting the database, and prompt you for confirmation.  Press **y** to continue.  The utility should complete successfully as shown here.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5f15214.PNG |
|  | Change to the /home/gpadmin/gpconfigs directory.  [gpadmin@mdw ~]$ **cd gpconfigs**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5f1e3f7.PNG |
|  | |  | | --- | | Using vi, edit the gpinitsystem\_configfile and replace the highlighted sections with the following:  Scroll down to the **OPTIONAL MIRROR PARAMETERS** section of the file and perform the following tasks:  1. Uncomment the lines shown in the snapshot.  2. Replace the declare line with the following:  **declare -a MIRROR\_DATA\_DIRECTORY=(/data/mirror)** |   C:\Users\cantot\AppData\Local\Temp\SNAGHTML61695b9.PNG |
|  | The final file appears as follows:  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6181582.PNG  This is a read-only file. Use :**wq!** to save the file. |
|  | Run the gpinitsystem utility to create a Greenplum Database system using the values defined in gpinitsystem\_config. The command is shown below:  gpinitsystem -c gpinitsystem\_config -h hostfile\_gpinitsystem \ -s *standby\_master\_hostname* -S   * *standby\_master\_hostname*is a variable and should be replaced with the standby hostname of your pod. * -s is an option used for a redundant system with a standby server. For more information on this utility, refer to Greenplum Database Installation Guide. * -Sis the option used to create a spread mirror configuration. For more information on this utility, refer to Greenplum Database Installation Guide.   [gpadmin@mdw gpconfigs]$ **gpinitsystem -c gpinitsystem\_config \ -h hostfile\_gpinitsystem -s smdw -S** |
|  | The utility displays the master, standby master, primary segments, and mirror segments configuration. It also asks you whether or not you want to continue with the Greenplum creation.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6198fff.PNG  Press **y** to continue. |
|  | This screen shows that the gpinitsystem completed with no errors.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML61a7925.PNG |
|  | Run the gpstateutility to verify the Greenplum instance status summary.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML61bbb23.PNG |
|  | Change to the home directory.  [gpadmin@mdw ~]$ **cd**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML678b359.PNG |

*End of Lab Exercise*

# Lab Using the PSQL Client and Greenplum Utilities

|  |  |
| --- | --- |
| **Purpose:** | In this lab you will learn how to connect to the database using psql, the command line client interface to the Greenplum Database. You will learn how to use psql to run SQL commands both in interactive mode and non-interactive mode. You will also learn how to view the help in psql and about psql meta-commands. |
|  | |
| **Tasks:** | Students perform the following tasks:   * Access the database with psql * Install and configure Greenplum Command Center |
|  | |
| **References:** | Module 3 – Greenplum Database Tools, Utilities, and Internals   * Lesson: Using the PSQL Client and Greenplum Utilities * Lesson: Pivotal Greenplum Command Center |

Lab 4: Task 1 – Access the Database with psql

| **Step** | **Action** |
| --- | --- |
|  | Start a psql session by connecting to the database, postgres:  [gpadmin@mdw ~]$ **psql postgres**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML67dbd86.PNG |
|  | The psql prompt displays the database you are connected to followed by =#, if you are logged in as the superuser account.  At the psql prompt, type the **\h** meta-command to see a list of available SQL commands:  postgres=# **\h**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6887b49.PNG |
|  | Use the \h meta-command to view the help for the SELECT SQL command only:  postgres=# **\h SELECT**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML688c9b6.PNG |
|  | At the psql prompt, type the \? meta-command to see a list of available psql meta-commands:  postgres=# **\?**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6892212.PNG |
|  | Run a SELECT statement on the gp\_segment\_configuration table. This system catalog table shows the entire master and segment instances in your array.  postgres=# **SELECT \* FROM gp\_segment\_configuration;**  Note the use of the semi-colon (;) at the end of the statement. This completes the SQL command, allowing pSQL to execute the command. If you do not include the semi-colon, but press return, your command will continue on the next line.  Your results should be similar to the following:  C:\Users\cantot\AppData\Local\Temp\SNAGHTML68a588d.PNG |
|  | Exit the psql session:  postgres=# **\q**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML68abb74.PNG |
|  | Execute the same SELECT statement you ran earlier, but this time, in non-interactive mode:  [gpadmin@mdw ~]$ **psql postgres -c 'SELECT \* FROM gp\_segment\_configuration;'**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML68c6440.PNG |
|  | Create a database with the same name as the user gpadmin. By creating a database with the same name as the current user, you will automatically be connected to this database if you do not specify the database name as part of the psql command line.   1. Connect to the postgres database: [gpadmin@mdw ~]$ **psql postgres** 2. Create a new database and name it **gpadmin**: postgres=# **create database gpadmin;** 3. Exit the database session: postgres=# **\q**   C:\Users\cantot\AppData\Local\Temp\SNAGHTML6c21325.PNG |
|  | Using vi, edit the .bash\_profile and add the PGDATABASE variable as shown below.  This variable contains your ***default login database name***. psql chooses a default database based on your username. The PGDATABASE environment variable takes precedence over this action, so if it is set, you will automatically be logged in to the database specified in the variable if you do not specify one on the command line.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6c42722.PNG  Run the command below to make the changes active immediately.  [gpadmin@mdw ~]$ **source .bash\_profile**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6c4846e.PNG |
|  | Copy the updated .bash\_profile file to the standby server smdw.  [gpadmin@mdw ~]$ **scp /home/gpadmin/.bash\_profile smdw:** |
|  | Log in to Greenplum with the default database of gpadmin.  [gpadmin@mdw ~]$ **psql**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6c73dfc.PNG |
|  | From within the database session, connect to the postgres database.  gpadmin=# **\c postgres**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6c7bc5f.PNG |
|  | Close your psql session.  postgres=# **\q**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6c8651c.PNG |

Lab 4: Task 2 – Install and Configure Pivotal Greenplum Command Center

| **Step** | **Action** |
| --- | --- |
|  | As root, connect to the master server, mdw, and change to the directory /rawdata/Binaries and list the contents of the directory:  [root@mdw ~]# **cd /rawdata/Binaries**  [root@mdw Binaries]# **ls** |
|  | Install the Greenplum Command Center binary to the master server, mdw, using the bash command.  [root@mdw Binaries]# **bash \ greenplum-cc-web-1.3.0.0-build-91-RHEL5-x86\_64.bin**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6d4253f.PNG  Hit **q** at any time to bypass the license agreement or space to read the next page of the license agreement. |
|  | Type **yes** and press **Enter** to accept the Greenplum Command Center license agreement.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6d58df0.PNG |
|  | Hit **Enter** to accept the default installation directory location:  /usr/local/greenplum-cc-web-1.3.0.0-build-91.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6d63085.PNG |
|  | Type **yes** and press **Enter** to install the Greenplum Command Center.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6d6a334.PNG |
|  | Type **yes** and press **Enter** to create the installation directory for the Greenplum Command Center.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6d715d4.PNG |
|  | Greenplum Command Center binaries will be installed into the default directory: /usr/local/greenplum-cc-web-1.3.0.0-build-91.  You will see the following output confirming that the binaries have been installed:  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6d7fc4c.PNG |
|  | Change the user and group permissions of Pivotal Greenplum Command Center directory to gpadmin**.**  [root@mdw Binaries]# **chown -R gpadmin:gpadmin \ /usr/local/greenplum-cc-web-1.3.0.0-build-91**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6dac89f.PNG |
|  | Change user to gpadmin.  [root@mdw Binaries]# **su - gpadmin**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6dcf20f.PNG |
|  | The installation directory contains a gpcc\_path.sh file with path and environment settings for the Console. Source the file as follows:  [gpadmin@mdw ~]$ **source /usr/local/greenplum-cc-web/gpcc\_path.sh**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6de78cb.PNG |
|  | Run the gpperfmon\_install utility to perform the following:   1. Create the Greenplum Command Center database (gpperfmon). 2. Creates the Greenplum Command Center superuser role (gpmon). 3. Configures Greenplum Database server to accept connections from the Greenplum Command Center superuser role (edits the pg\_hba.conf and .pgpass files). 4. Sets the Greenplum Command Center server configuration parameters in the Greenplum Database server postgresql.conf files.   [gpadmin@mdw ~]$ **gpperfmon\_install \ --enable --password changeme --port 5432**  You will see the following output to the console: |
|  | C:\Users\cantot\AppData\Local\Temp\SNAGHTML6e55fa0.PNG |
|  | Modify the .bash\_profile file and add an entry for the GPPERFMONHOME variable. This variable, which specifies the home directory for Greenplum Command Center, should be set to **/usr/local/greenplum-cc-web**.  Add the following to the file:  **GPPERFMONHOME=/usr/local/greenplum-cc-web**  **export GPPERFMONHOME**  **source $GPPERFMONHOME/gpcc\_path.sh**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML22436de2.PNG |
|  | Source /home/gpadmin.bash\_profile  [gpadmin@mdw ~]$ **source /home/gpadmin/.bash\_profile** |
|  | Add a record to pg\_hba.conf to allow Command Center to connect to localhost through the IPv6 address ::1.  [gpadmin@mdw ~]$ **echo “host     all    gpmon   ::1/128     md5” >> \ /data/master/gpseg-1/pg\_hba.conf**  **NOTE**: Tab characters are not supported in the pg\_hba.conf file. You must use spaces between the record entries. |
|  | Restart the Greenplum Database cluster to enaba -rle gpperfmon.  [gpadmin@mdw ~]$ **gpstop -a -r**  The gpstop command may take some time to stop the database.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6f39cfe.PNG |
|  | After the Greenplum Database Cluster has restarted, execute the following command:  [gpadmin@mdw ~]$ **gpcmdr --setup**  You will be prompted for the instance name:  Type in **gp1** and press **Enter**.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML70a7811.PNG |
|  | You will be prompted with the question, Is the master host remote?  Press **Enter** to accept the default answer of No.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML70bc43d.PNG |
|  | You will be prompted for the display name of the instance:  Type in **traindb** and press **Enter**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML70c719b.PNG |
|  | You will be prompted for the port number of the database. Press **Enter** to accept the default port number of 5432. A message will appear indicating that the instance schema is being created.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML70d2b87.PNG |
|  | You will be prompted for the web server port. Press **Enter** to accept the default port of 28080.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML70e1134.PNG |
|  | You will be prompted to enable SSL for the Web API. Type **Y** and press **Enter** to confirm SSL.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML70ed82a.PNG |
|  | You will be prompted with a series of questions for your certificate. Enter the following information for the certificate:   * Country name: **US** * State or province name: **CA** * Locality name: **Palo Alto** * Organization name: **EdServices** * Organizational Unit: **PivotalEdu** * Common name: **mdw** * Email address: **gpadmin@mdw**   C:\Users\cantot\AppData\Local\Temp\SNAGHTML710170d.PNG |
|  | When prompted to enable IPV6, press the **ENTER** key to accept the default No.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML71110d6.PNG |
|  | When prompted to enable cross site request forgery protection, press **Enter** to accept the default of no.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML7128f96.PNG |
|  | When prompted to copy the instance to the standby server, type **N** and press **Enter**.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML718fc5d.PNG  The configuration process will complete, displaying the following:  C:\Users\cantot\AppData\Local\Temp\SNAGHTML7196eed.PNG |
|  | Start the Greenplum Command Center web service for the instance you created using the  gpcmdr --start gp1 command.  [gpadmin@mdw ~]$ **gpcmdr --start gp1**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML72aab2.PNG |
|  | Using a browser of choice, connect to the URL, **https://172.16.1.11:28080**.  Depending on which browser you are using you may receive a “Warning” or “Exception” that the connection is “Insecure”. Since Greenplum is running on the same machine simply accept the “Exception” and continue. |
|  | The login screen will now be displayed. Log in with the following credentials:   * Username: **gpadmin** * Password: **changeme**     Click **Login** to proceed.  **Note:** The gpadmin user is a trusted account within the Greenplum Database. You can further secure the environment by requiring that the gpadmin user account enter a MD5 or SHA-256 encrypted password. |
|  | The dashboard will be displayed and will show the current system performance information. |

Lab 4: Task 3 – Navigating Pivotal Greenplum Command Center

| **Step** | **Action** |
| --- | --- |
|  | In this task, you will explore the Pivotal Greenplum Command Center environment, managing and monitoring activities within the environment.  The Dashboard provides an overview of the environment.  Collect the following information:   * What is the health of the database? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ * How many segments are visible within the environment and what is their status? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Click **More »** in the Segment Health panel. The Administration tab should now be displayed with the Segment Health panel displayed.  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLdb084d.PNG  Are all of the segments and mirrors in their preferred roles? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | From the left panel, click Database Usage Report. Click Generate Report to run a report highlighting how the database is being used. The results will be displayed in the Summary panel.  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLe00947.PNG  Once the database is populated, this provides a graphical method of seeing the structure of the database, including the type of tables and the different databases available in the environment.  Based on the report, are there any databases listed in your environment? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | From the left panel, click Storage Monitoring. The page displays a usage summary for your environment.  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLe3f7ab.PNG  Information on the GP Master is displayed by default. This includes all masters, the primary and standby masters. Use the panel below the Disk Usage Summary to answer the following:   * What is the data directory for the master and standby servers? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ * What is the percentage of data used on sdw1? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | From the left panel, click Database Admin. This displays a panel that allows you to change the state of the database.  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLe93d81.PNG |
|  | Restart the database in Smart mode. If there are any open connections to the database, a dialog box will be displayed asking if you wish to continue the restart with the **fast** option.    Click **OK** to proceed.  You should have at least one connection to the database. Where is that connection coming from?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Enter the password for the gpadmin account and click **Restart**. |
|  | You will be asked to confirm, with a dialog warning you that you will end your session.    Click **OK** to proceed.  You should see the following as the database is restarted. |
|  | Once the database has restarted, click **OK** to return to the login screen to log back in. |
|  | Log into Pivotal Command Center with the gpadmin user account. |

*End of Lab Exercise*

# Lab Greenplum Database Server Configuration

|  |  |
| --- | --- |
| **Purpose:** | In this lab you will learn how to set configuration parameters in postgresql.conf and use the SHOW command in psql to examine the current values of configuration parameters. You will also learn how to set Greenplum global parameters on a per-session basis using the SET command in psql. |
|  | |
| **Tasks:** | Students perform the following tasks:   * + Set Configuration Parameters   + Configure host-based authentication |
|  | |
| **References:** | Module 3 – Greenplum Database Tools, Utilities, and Internals   * Lesson: Greenplum Database Server Configuration |

Lab 5: Task 1 – Set Configuration Parameters

| **Step** | **Action** |
| --- | --- |
|  | Change to the data directory of your Greenplum Database master instance:  [gpadmin@mdw ~]$ **cd $MASTER\_DATA\_DIRECTORY**  [gpadmin@mdw gpseg-1]$ **ls**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML12c6b84.PNG |
|  | In a text editor, edit the postgresql.conf file.  [gpadmin@mdw gpseg-1]$ **vi postgresql.conf**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML12d15f6.PNG |
|  | Find the configuration parameter named log\_min\_messages. This parameter controls the level of detail written to the log files.  Uncomment this line by removing the **#** preceding it. Change the value from warning to **info**. **log\_min\_messages = info**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML13330c6.PNG  Save your changes and exit vi. |
|  | The postgresql.conf file for the master instance must be reloaded so that the changes are picked up by the Greenplum Database. Use the **gpstop** utility to complete this task:  [gpadmin@mdw gpseg-1]$ **gpstop -u**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML12f7841.PNG |
|  | Once the change has been loaded, start a psql session and check the value using the SHOW command to verify the log level was changed to **info**:  [gpadmin@mdw gpseg-1]$ **psql postgres**  postgres=# **SHOW log\_min\_messages;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1340830.PNG |
|  | While still in psql, change the value of log\_min\_messages to debug1 using the SET command. This sets the parameter for the current session only.  postgres=# **SET log\_min\_messages TO debug1;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML135e593.PNG |
|  | Check the value using the SHOW command to make sure it was changed to debug1:  postgres=# **SHOW log\_min\_messages;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1368348.PNG |
|  | Exit the current psql session and start a new session.  Run the SHOW command again. The value should be info again, the value you set in postgresql.conf  postgres=# **\q**  [gpadmin@mdw gpseg-1]$ **psql postgres**  postgres=# **SHOW log\_min\_messages;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1380cf1.PNG |
|  | Exit psql:  postgres=# **\q** |
|  | Using the gpconfig command, list the values of the log\_min\_messages parameter.  [gpadmin@mdw gpseg-1]$ **gpconfig -s log\_min\_messages**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1bf7834.PNG |
|  | Use the gpconfig command to determine if the same command can be used to set the log\_min\_messages parameter.  [gpadmin@mdw gpseg-1]$ **gpconfig -l | grep log\_min\_messages**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1c67a94.PNG |
|  | Using the gpconfig command, modify the log\_min\_messages parameter once more to change it to warning.  [gpadmin@mdw gpseg-1]$ **gpconfig -c log\_min\_messages -v warning**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1d911f3.PNG |
|  | Using gpstop, reread the postgresql.conf file.  [gpadmin@mdw gpseg-1]$ **gpstop -u**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1db838a.PNG |
|  | Use gpconfig to verify the log\_min\_messages parameter has been updated on all servers.  [gpadmin@mdw gpseg-1]$ **gpconfig -s log\_min\_messages**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1ddf1e6.PNG |
|  | Using the gpconfig command, remove the changes that you made to the log\_min\_messages parameter.  [gpadmin@mdw gpseg-1]$ **gpconfig -r log\_min\_messages**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1df6bc6.PNG |
|  | Reload the postgresql.conf file.  [gpadmin@mdw gpseg-1]$ **gpstop -u**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1e08aea.PNG |
|  | Verify the changes have been reset.  [gpadmin@mdw gpseg-1]$ **gpconfig -s log\_min\_messages**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1ddf1e6.PNG  Is this the expected value? Why or why not?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | **Summary**  Setting server configuration parameters can be done at the system level by editing the postgresql.conf configuration file on the Greenplum master. The preferred method for editing some parameters is with the gpconfig command. The command, gpconfig -l lists all the parameters you can change with gpconfig. You must either reload the master postgresql.conf file with the gpstop –u command or restart the Greenplum Database with the gpstop -r before changes are visible.  Note that some parameters are global and some are not. Global parameters need only be set on the master and the value is passed on to the segments. The non-global parameters must be set in the postgresql.conf on the master and on all segments as well and require a system restart.  You can also set global parameters on a per-session basis using the SET command. Note that it is also possible to set parameters at the database level (using ALTER DATABASE SET) or at the role level (ALTER ROLE SET).  To examine the current setting of configuration parameters, use the SHOW command. You can also use the gpconfig -s command to list the values on all servers in the cluster. SHOW ALL will show the values for all parameters. You will have to page through the commands or type q to stop.  Optional steps:   1. Connect to the postgres database: $ **psql postgres** 2. Issue the **SHOW ALL** command: postgres=# **SHOW ALL;** 3. Exit **psql**:postgres=# **\q** |

Lab 5: Task 2 – Configure Host-Based Authentication

| **Step** | **Action** |
| --- | --- |
|  | If not already there, change to the directory represented by the variable, $MASTER\_DATA\_DIRECTORY**,** to perform this exercise. Using vi, edit the pg\_hba.conf file.  [gpadmin@mdw gpseg-1]$ **cd $MASTER\_DATA\_DIRECTORY**  [gpadmin@mdw gpseg-1]$ **vi pg\_hba.conf** |
|  | In follow on labs, you will create a database called faa and you will add a database role for yourself. Add the following line at the bottom of pg\_hba.conf, replacing <your\_name> with your first name:  **local faa <*your\_name*> trust**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML13e4b38.PNG  Save your changes and exit vi.  Before proceeding to the next step:   1. Verify that all characters entered are lowercase. 2. Do not use tabs to separate each field in the pg\_hba.conf file. Instead, use spaces. |
|  | Reload the configuration for the changes to take effect:  [gpadmin@mdw gpseg-1]$ **gpstop -u**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML13f153b.PNG |
|  | **Summary**  The pg\_hba.conf file controls the hosts and users that can connect to the Greenplum Database, the databases they can access, and how users are authenticated.  After a Greenplum Database system is first initialized, only the Greenplum superuser, the user who ran gpinitsystem, is allowed to connect from the local host. If you want to allow other users or client machines to connect, or change the default means of authentication, you must edit the pg\_hba.conf file of the master instance.  It is often necessary to give access with the addition of new databases or when requesting remote access through a query editor such as pgAdmin III.  **Note:** You must reload the configuration (or restart the Greenplum Database) for changes to pg\_hba.conf to take effect. |

*End of Lab Exercise*

This page intentionally left blank

# Lab Database Internals

|  |  |
| --- | --- |
| **Purpose:** | Each instance in the Greenplum Database, the master and each segment, has a data directory where database files are stored on disk. In this lab, you examine the database files on disk and determine which database objects they correspond to by looking up their OID (object identifier) numbers in the system catalog tables.  You will examine the database server processes running on a Greenplum Database host. |
|  | |
| **Tasks:** | Students perform the following tasks:   * + Explore database files on disk   + Examine server processes |
|  | |
| **References:** | Module 3 – Greenplum Database Tools, Utilities, and Internals   * Lesson: Database Internals |

Lab 6: Task 1 – Explore Database Files on Disk

| **Step** | **Action** |
| --- | --- |
|  | You should be logged as gpadmin at MASTER\_DATA\_DIRECTORY. If you are not at the master data directory cd to it.  [gpadmin@mdw ~]$ **cd $MASTER\_DATA\_DIRECTORY**  [gpadmin@mdw gpseg-1]$ **ls**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML140ad55.PNG  You should see several directories. |
|  | Create a database called names.  [gpadmin@mdw gpseg-1]$ **createdb names**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1411b43.PNG  **Note:** The createdb command is a Greenplum utility that is executed from the command line. The equivalent SQL command is  CREATE DATABASE names; |
|  | Change to the base subdirectory. This is where database files are stored. You should see a subdirectory for each database that exists in your Greenplum Database system:  [gpadmin@mdw gpseg-1]$ **cd base**  [gpadmin@mdw base]$ **ls**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1434c80.PNG  How many subdirectories are located in the base subdirectory? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Find out which database each directory corresponds to by looking up the OID number in the system catalog table, pg\_database:  [gpadmin@mdw base]$ **psql -c "SELECT datname, oid FROM pg\_database;"**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML145cff2.PNG **Note:** Your output may differ from the screenshot shown here.  Record the OID of the names database in the space provided. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Using the SQL statement provided, create a schema and tables in the names database:  [gpadmin@mdw base]$ **psql -d names**  names=# **CREATE SCHEMA baby;**  names=# **CREATE TABLE baby.names (id int PRIMARY KEY,name varchar(30), origin varchar(30), meaning text);**  names=# **CREATE TABLE baby.rank (id int, rank int, year int, gender char(1), count int ) DISTRIBUTED BY (id, gender, year);**  names=# **\q**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML147323d.PNG  You will examine the syntax in greater detail in upcoming lessons and labs. |
|  | Change to the namesdatabase directory on disk and examine the files in that subdirectory. You must know the OID of the names database.  There are several files, identified by OID, that correspond to database objects. View the contents of the names database represented by its OID with the following commands:  [gpadmin@mdw base]$ **cd <oid\_of\_names\_db>** [gpadmin@mdw base]$ **ls**  In this example, we use the OID for the names database obtained earlier in the lab.  [gpadmin@mdw base]$ **cd 35957**  [gpadmin@mdw 35957]$ **pwd**  [gpadmin@mdw 35957]$ **ls**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15cee6a.PNG |
|  | Discover which OID file corresponds to the rank table by looking it up in the pg\_class table:  [gpadmin@mdw 35957]$ **psql names -c "SELECT oid FROM pg\_class WHERE relname='rank';"**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15f3f47.PNG  Which OID corresponds to the rank table? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | **Summary**  Each master and segment instance in the Greenplum Database has a data directory where database files are stored on disk. Each database object on disk can be identified by looking up its OID in the system catalog tables. Database objects in the Greenplum Database are distributed across the segment instances. The OIDs are consistent across all instances in the array. |

Lab 6: Task 2 – Examine Server Processes

| **Step** | **Action** |
| --- | --- |
|  | You should be logged in to the master server, mdw, as gpadmin**.** Change to the $MASTER\_DATA\_DIRECTORY directory. Find all of the postgres processes running on the system:  [gpadmin@mdw 35957]$ **cd $MASTER\_DATA\_DIRECTORY**  [gpadmin@mdw gpseg-1]$ **ps ax | grep postgres**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML19fa77d.PNG  How many are there? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Can you identify the database listener process, also known as the postmaster? |
|  | Connect directly to a segment instance using psql and a segment instance port:  [gpadmin@mdw gpseg-1]$ **psql names -p 50001**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a0a9be.PNG  What happened? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Using the gpconfig command, view the defined ports for the database by examining the **port** parameter.  [gpadmin@mdw gpseg-1]$ **gpconfig -s port**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f37fa4.PNG |
|  | Attempt to connect to the one of the segment ports listed in the previous step.  [gpadmin@mdw gpseg-1]$ **psql names -p 40000**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f6830e.PNG  Where you able to connect? Why or why not?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | **Summary**  The postgres database server process, known as the postmaster in prior releases accepts client connections. In the Greenplum Database, only the postgres listener process of the master instance can accept user client connections. The segments only allow connections from the master. The database listener process has several sub postgres processes running at any given time. Sub postgres processes are spawned as needed to handle client requests. In the Greenplum Database, the ps output for a postgres client process will have similar information as found in the Greenplum Database server logs:  postgres: <user><database><host><con#><seg#><cmd#><slice#|status> |

*End of Lab Exercise*

This page intentionally left blank

# Lab Data Definition Language

|  |  |
| --- | --- |
| **Purpose:** | In this lab, you will learn how to manage databases and schemas in the Greenplum Database. You will create a database called faa and a schema called faadata. You will also set the schema search path and learn how to verify which schema you are in.  You will also create some tables in the Greenplum Database and learn how the Greenplum distribution key for a table is chosen. |
|  | |
| **Tasks:** | Students perform the following tasks:   * + Create databases and schemas   + Create tables   + View indexes and schemas |
|  | |
| **References:** | Module 4 – Defining and Securing the User Database   * Lesson: Data Definition Language |

Lab 7: Task 1 – Create Databases and Schemas

| **Step** | **Action** |
| --- | --- |
|  | Log in as gpadmin, create a new database called faa using the createdb Greenplum client:  [gpadmin@mdw ~]$ **createdb faa**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5ed07fa.PNG |
|  | Set the faa database to be the default database you connect to when you start a psql session without specifying the database name:  [gpadmin@mdw ~]$ **export PGDATABASE=faa**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5ed7d68.PNG |
|  | Verify that the **PGDATABASE** environment variable is set correctly:  [gpadmin@mdw ~]$ **echo $PGDATABASE**  It should now be set to faa. C:\Users\cantot\AppData\Local\Temp\SNAGHTML5ef04df.PNG |
|  | Start a psql session and connect to the faa database. If the PGDATABASE environment variable is set correctly, you do not need to supply the database name:  [gpadmin@mdw ~]$ **psql**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5ef5fe9.PNG |
|  | At the psql prompt, list all the databases in the system with the **\l** meta-command.  faa=# **\l**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5efdd04.PNG  Which ones do you see? Is the faa database there? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | At the psql prompt, create a schema named faadatausing the CREATE SCHEMA SQL command:  faa=# **create schema faadata;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5f1b2aa.PNG |
|  | Change the search path on the database**,** faaso that the new faadata schema is the default schema:  faa=# **ALTER DATABASE faa SET search\_path TO faadata, public, pg\_catalog;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5f24e8b.PNG |
|  | The change to the search path will not become visible until the next client connection. Use the \c meta-command to reconnect to the faa database:  faa=# **\c faa**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5f2a39d.PNG |
|  | Run the current\_schema() function to verify that you are indeed in the faadata schema:  faa=# **SELECT current\_schema();**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5f316f8.PNG |
|  | Examine the value of the search\_path configuration parameter to verify it is correct:  faa=# **SHOW search\_path;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5f37849.PNG |
|  | Use the \dn meta-command to list the schemas in the database:  faa=# **\dn**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5f3e37a.PNG  Which ones do you see? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Which ones are for system-level objects and which ones are for user-created objects? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | **Summary**  You can create or drop databases using the CREATE DATABASE or DROP DATABASE commands or the createdb or dropdb client programs respectively.  By default, every newly created database has a schema named public, which is where objects are created by default.  If you create your own schemas and do not want to use qualified names all the time, you should set the search\_path parameter to ensure your schema is first in the search path. You can use the ALTER DATABASE command to set the schema search path for the database.  You can also use the ALTER ROLE command to set the schema search path for a particular role. |

Lab 7: Task 2 – Create Tables

| **Step** | **Action** |
| --- | --- |
|  | You will create tables in the faa Greenplum database and learn how the Greenplum distribution key for a table is chosen.  For the purpose of this lab exercises, the faa database will have two tables:   * The test\_table table will act as a fact data. * The test\_table2 table will act as a dimension table.   At the psql prompt, create a new table with the following definition:  faa=# **CREATE TABLE test\_table (**  **id int PRIMARY KEY,**  **name varchar(30) NOT NULL,**  **origin varchar(30) NOT NULL,**  **meaning text**  **);**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5f54ac4.PNG  Note that the table has a PRIMARY KEY and there is no explicit DISTRIBUTED BY clause given. The primary key is automatically chosen as the distribution key in this case. |
|  | Create a table with the following definition and explicitly declare a distribution key using a DISTRIBUTED BY clause:  faa=# **CREATE TABLE test\_table2 (**  **id int,**  **rank int,**  **year int,**  **count int)**  **DISTRIBUTED BY (id, year);**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5f5b78a.PNG |
|  | Why is this distribution key a good choice for this table? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  What would be the distribution key if you left out the DISTRIBUTED BY clause? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Would that be a good distribution key for this table? Is that a unique key? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Use the \dt meta-command to list the tables in the database:  faa=# **\dt**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5f61a42.PNG  Which ones do you see? Are the test\_table and test\_table2 tables there? |
|  | Exit psql:  faa=# **\q** |
|  | **Summary**  When creating the table, there is an additional clause to declare the Greenplum distribution key column(s).  If a DISTRIBUTED BY clause is not supplied, then either the PRIMARY KEY, if the table has one, or the first column of the table will be used. This may or may not be the desirable distribution key.  To ensure an even distribution of data, choose a distribution key that is unique for each record. If that is not possible, the column(s) with the lowest selectivity can also be chosen. If a good choice of distribution columns is not available, choose DISTRIBUTED RANDOMLY as the distribution key. |

Lab 7: Task 3 – View Indexes and Schemas

| **Step** | **Action** |
| --- | --- |
|  | Start a psql session:  [gpadmin@mdw ~]$ **psql**  Verify you are in the faa database. If not, issue the following meta-command: **\c** faa. |
|  | At the psql prompt, create a new view on the test tables:  faa=# **CREATE VIEW test\_view AS**  **SELECT test\_table.id**  **FROM test\_table, test\_table2**  **WHERE test\_table.id < 11 AND test\_table.id = test\_table2.id;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5f7ddb0.PNG |
|  | Confirm the view definition is correct:  faa=# **\d+ test\_view**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5f840d5.PNG |
|  | Create an index on the id column of the test\_table2 table:  faa=# **CREATE INDEX test\_index ON test\_table2 (id);**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5f8a967.PNG |
|  | Confirm the index was created:  faa=# **\di**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5f8f767.PNG |
|  | Create a sequence table that you can use to assign unique ids when inserting new records into the test\_table table. The id value in the test\_table table is currently 1, so the sequence starts at 1:  faa=# **CREATE SEQUENCE test\_table\_seq START 1;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5f993e4.PNG |
|  | Examine the sequence table you just created: faa=# **SELECT \* FROM test\_table\_seq;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5f9f297.PNG |
|  | Exit psql: faa=# **\q** |
|  | **Summary**  Views allow you to store frequently used queries and then access them in SELECT statements as if they were a regular table. You created a view that selects the top ten baby names from your database.  Indexes are not always the performance enhancer in the Greenplum Database as they are in traditional database management systems. In some cases indexes can improve query performance, and in some cases indexes have no effect or can slightly degrade performance. You have added an index on the rank table id column, since that column will be used often in your queries to join with the names table. When you get to query profiling, you will determine if this index is indeed being utilized.  Sequences are used to generate numbers, helpful for incrementing unique id columns such as the id column of your names table. You can use the sequence when you insert new names into this table to generate a unique id number that won’t conflict with the other id numbers already being used. |

*End of Lab Exercise*

# Lab Data Manipulation Language and Data Query

|  |  |
| --- | --- |
| **Purpose:** | In this lab, you will familiarize yourself with the INSERT, UPDATE, and DELETE SQL commands. |
|  | |
| **Tasks:** | Students perform the following task: Insert, update, and delete records. |
|  | |
| **References:** | Module 4 – Defining and Securing the User Database   * Lesson: Data Manipulation Language and Data Query Language |

Lab 8: Task 1 – Insert, Update, and Delete Records

| **Step** | **Action** |
| --- | --- |
|  | Connect to the faa database as the gpadmin user, if not already connected:  [gpadmin@mdw ~]$ **psql faa**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6264eac.PNG  You can also specify the database using the -d option followed by the database name. |
|  | At the psql prompt, add a new record to the test\_tabletable.  faa=# **INSERT INTO test\_table VALUES**  **(nextval('test\_table\_seq'), 'Esme', 'French', 'to love');**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML627446f.PNG |
|  | Check to see that the record you created is there:  faa=# **SELECT \* from test\_table WHERE name='Esme';**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML627a41b.PNG |
|  | Update the record to change the name column to Sophie for any instances where the name column is Esme:  faa=# **UPDATE test\_table SET name='Sophie' WHERE name='Esme';**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML628207b.PNG |
|  | Check to see that the record was changed:  faa=# **SELECT \* from test\_table WHERE name='Sophie';**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML62876c4.PNG |
|  | Delete the record. Do not forget the WHERE clause or you will delete all of the rows:  faa=# **DELETE from test\_table WHERE name='Sophie';**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML628cb88.PNG |
|  | Check to see that the record was deleted:  faa=# **SELECT \* from test\_table WHERE name='Sophie';**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML629365b.PNG  Exit your psql session.  faa=# **\q** |

Lab 8: Task 2 – Accessing Data to Generate Reports

| **Step** | **Action** |
| --- | --- |
|  | In this task, you use a variety of statements to generate a report. The report identifies the least popular baby names, across boys and girls, from data collected. |
|  | Connect to the names database as the gpadmin user, if not already connected:  [gpadmin@mdw ~]$ **psql names** |
|  | Update the search\_path for the names database so that you can easily access the tables from the names database without needing to specify the schema name.  names=# **alter database names set search\_path to baby, public, pg\_catalog;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML64f71d6.PNG |
|  | Reconnect to the database to access the settings.  names=# **\c names**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6530f6c.PNG |
|  | Identify the tables in this database and use COUNT to verify the tables have no rows.  names=# **\dt**  names=# **select count(\*) from names;**  names=# **select count(\*) from rank;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML68befeb.PNG |
|  | Using the psql copy command, copy data from local files into the rank table. The data will be copied from the CSV file, /home/gp/sql/load\_files/boys, and includes a header line. This command behaves like the SQL COPY command, which will be covered later in the course. For now, you are populating tables with values to be manipulated and accessed.  names=# **\copy rank from /home/gp/sql/load\_files/boys header delimiter as ',' csv**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML65b05fa.PNG |
|  | Using COUNT, verify data has now been loaded into the rank table.  names=# **select COUNT(\*) from rank;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML68f40b9.PNG |
|  | Using the syntax shown earlier, load data from the files noted to the correlating tables.  Use the up arrow (↑) to scroll through your PSQL buffer until you find the \copy command and change the line as needed. Use the left (←) and right (→) arrows to move through the command to the required position.   |  |  | | --- | --- | | File | Table | | /home/gp/sql/load\_files/girls | rank | | /home/gp/sql/load\_files/name\_ids | names |   names=# **\copy rank from /home/gp/sql/load\_files/girls header delimiter as ',' csv**  names=# **\copy names from /home/gp/sql/load\_files/name\_ids header delimiter as ',' csv**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6990984.PNG |
|  | Use COUNT to verify that the rows have been loaded into the respective tables.  names=# **select COUNT(\*) from rank;**  names=# **select COUNT(\*) from names;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML69a9732.PNG |
|  | The rank table highlights the number of occurrences of babies given a specific name. Each row indicates the ID for the name, the name’s ranking or weight, the year the information was collected, and the number of babies given the specific name. The names table associates the ID with the name and provides the origin and local meaning of the name. While the rank can be used to determine the least popular name as defined by the rank, this task will have you use the aggregate functions against the count associated with the names.  Use the MIN function to find the lowest count of a name for each year. Remember, MIN is an aggregate function and requires the use of GROUP BY.  names=# **select min(count), year from rank group by year;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6c55a52.PNG |
|  | Examine the rank table to list the ID(s) that had the lowest count for the year 2004. Use the information returned from the previous example to complete the request.  names=# **select id from rank where year=2004 and count=164;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6ca7ee3.PNG |
|  | This two-step process can easily be changed to a single step and would allow us to collect the information for all years involved. Use the WITH clause to define a common table expression that allows you to return the ID, year, and count for the lowest assigned name for each year.  names=# **WITH min\_count AS**  **(select min(count) as mcount, year from rank group by year)**  **select rank.id, rank.year, rank.count from min\_count, rank**  **where min\_count.year=rank.year and min\_count.mcount=rank.count;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6cfcaa2.PNG  Tip: If you are familiar with the vi editor, you can edit the previous PSQL statement using the \e command. It will take you into the editor and allow you to create or modify your statement. When done modifying your statement, use :wq! to save and execute the statement. Prefer Emacs or VIM, exit psql, update the EDITOR variable to point to your preferred editor, and re-enter PSQL. For example, to set the editor to Emacs in this environment, execute the following in the shell or add it to your .bash\_profile:  **export EDITOR=/usr/bin/emacs** |
|  | If this will be a frequently accessed statement, it would be best to save it as a view. Create a view called min\_count\_vw that contains the common table expression you defined in the previous step.  names=# **CREATE VIEW min\_count\_vw AS**  **WITH min\_count AS**  **(select min(count) as mcount, year from rank group by year)**  **select rank.id, rank.year, rank.count from min\_count, rank**  **where min\_count.year=rank.year and min\_count.mcount=rank.count;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6d562ac.PNG |
|  | Using the view you created, list the least assigned baby names for the years collected. Make the report easier to read by sorting the results by year.  names=# **select name, year**  **from min\_count\_vw, names**  **where names.id=min\_count\_vw.id**  **order by year;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML6e2dd38.PNG |
|  | **Summary**  If you have many rows to insert, consider using COPY or external tables instead of INSERT.  UPDATE is used to change one or more column values of an existing row. DELETE is used to delete rows from a table based on some qualifying condition.  An unqualified delete on a table will delete all rows. A more efficient way to delete all rows from a table is the TRUNCATE command.  A variety of tools are available to help generate information for reports. A portion of this lab exercise focused on using some aggregate functions to collect the information that was needed. Views and common table expressions let you simplify more complex statements and make these statements available for future use. |

*End of Lab Exercise*

# Lab Roles, Privileges, and Resources

|  |  |
| --- | --- |
| **Purpose:** | In this lab you will learn how to create roles that are users and roles that are groups.  User-level roles can log in to a database. Group-level roles are useful for managing permissions and access privileges to database objects.  You will learn how to grant privileges on database objects to a group-level role. Members, or user-level roles, of the group inherit those access privileges.  You will create a resource queue and assign a user role to this resource queue. The user name you use is the name you added in the pg\_hba.conf file, your first name. For testing purposes, the resource queue you create will have an ACTIVE THRESHOLD of 1, meaning that only one active statement submitted through the resource queue will be allowed to run at any given time. |
|  | |
| **Tasks:** | Students perform the following tasks:   * + Manage user and group roles   + Manage resource queues |
|  | |
| **References:** | Module 4 – Defining and Securing the User Database   * Lesson: Roles, Privileges, and Resources |

Lab 9: Task 1 – Manage User and Group Roles

| **Step** | **Action** |
| --- | --- |
|  | Start a psql session: [gpadmin@mdw ~]$ **psql**  If you are not in the faa database, issue the following meta-command to connect to the database: **\c faa**. |
|  | Create a role named admin and give it appropriate permission attributes for an administrator group-level role using the following command  faa=# **CREATE ROLE admin WITH CREATEDB CREATEROLE;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbc15e23.PNG |
|  | Create a user login role for yourself and give it appropriate permission attributes: faa=# **CREATE ROLE <your\_name> WITH LOGIN;**  Replace **<*your\_name*>** with the name you used in Lab 5: Greenplum Database Server Configuration.  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbc207ca.PNG |
|  | Add the user role you just created as a member of the admin role using the GRANT command: faa=# **GRANT admin TO <your\_name>;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbc25eb0.PNG |
|  | Confirm that your roles are configured as expected by using the \du meta-command in psql: faa=# **\du**  This shows a list of all roles in the system.  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbc2abc6.PNG |
|  | Change to the user role you just created using the SET ROLE command: faa=# **SET ROLE TO <*your\_name*>;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbc2fe0a.PNG  Note that your prompt has changed to reflect the level of the user you have connected into the database with. |
|  | Run a query as this role. Select all columns from the test\_table table: faa=> **SELECT \* from test\_table;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbc37a5a.PNG  The command did not succeed. Why not? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Exit psql: faa=> **\q** |
|  | Set the PGDATABASE variable to faa.  [gpadmin@mdw ~]$ **export PGDATABASE=faa** |
|  | Log in to the database as **gpadmin**.  [gpadmin@mdw ~]$ **psql**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbc4815c.PNG |
|  | Grant privileges to the faa database, faadata schema, and test\_table and test\_table2 tables to the admin role. Grant all privileges on the faa database:  faa=# **GRANT ALL ON DATABASE faa TO admin WITH GRANT OPTION;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbc50589.PNG |
|  | Grant all privileges on the schema, faadata using the GRANT OPTION clause:  faa=# **GRANT ALL ON SCHEMA faadata TO admin WITH GRANT OPTION;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbc56b3e.PNG |
|  | Grant all privileges to admin on the tables, test\_table and test\_table2 with the GRANT OPTION clause:  faa=# **GRANT ALL ON TABLE test\_table, test\_table2 TO admin WITH GRANT OPTION;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbc5c426.PNG |
|  | Exit psql:  faa=# **\q** |
|  | Start a psql session as the user role you created earlier and connect to the faa database:  [gpadmin@mdw ~]$ **psql -U student faa**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbc636d6.PNG |
|  | Run the SELECT query you ran earlier to test access to the tables. You should now have permissions to see this view (and its schema):  faa=> **SELECT \* from test\_table;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbc696c0.PNG |
|  | Exit psql:  faa=> **\q** |
|  | **Summary**  Greenplum manages database access permissions using the concept of roles. The concept of roles subsumes the concepts of users and groups. A role can be a database user, a group, or both. Roles can own database objects (for example, tables) and can assign privileges on those objects to other roles to control access to the objects. Roles can be members of other roles, thus a member role can inherit the attributes and privileges of its parent role.  Note that if you are using table inheritance and partitioning, table privileges are not passed down from parent tables to child tables. You must explicitly set privileges on each child table. If you use the Greenplum management utilities to partition a table, the utility takes care of passing down the table permissions for you.  When you create a new login role, you must be sure that the pg\_hba.conf configuration file of the master allows the role to connect to the Greenplum Database. Otherwise the role will be denied access. |

Lab 9: Task 2 – Manage Resource Queues

| **Step** | **Action** |
| --- | --- |
|  | At the psql prompt, create a resource queue named adhoc and give it an ACTIVE THRESHOLD limit of 1.  Connect to the faa database as gpadmin:  [gpadmin@mdw ~]$ **psql** |
|  | Create the resource queue, adhoc, with ACTIVE THRESHOLD equal to 1:  faa=# **CREATE RESOURCE QUEUE adhoc ACTIVE THRESHOLD 1;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbc9c6b5.PNG |
|  | Confirm the resource queue was created correctly by checking the pg\_resqueue system table:  faa=# **SELECT \* from pg\_resqueue;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbca3f9c.PNG  Note: The pg\_default resource queue is the default queue to which all roles are assigned if the resource queue is not specified when the role is created. |
|  | Assign the <your\_name> user-level role that you created to the adhoc resource queue:  faa=# **ALTER ROLE <your\_name> RESOURCE QUEUE adhoc;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbcb3cfd.PNG |
|  | Confirm that the role was assigned the adhoc resource queue by querying the pg\_resqueue and pg\_roles system tables:  faa=# **SELECT rolname, rsqname FROM pg\_roles AS r, pg\_resqueue AS q WHERE r.rolresqueue=q.oid;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbcbdd51.PNG |
|  | Exit the psql session:  faa=# **\q** |
|  | Start a new psql session as the <*your\_name*> user-level role:  [gpadmin@mdw ~]$ **psql -U student**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbcc6564.PNG |
|  | Verify that the resource queue is working.  To hold a query open, open a cursor within a transaction. This action holds the one active query slot you are allowed in the adhoc resource queue. The cursor will remain open until the transaction is closed, which will give you the time required to test the resource queue’s limits:  faa=> **BEGIN;** faa=> **DECLARE rqtest CURSOR FOR SELECT \* FROM test\_table;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbccfa62.PNG |
|  | Leave the current session open, and open a new terminal session to the master server. Connect as root and switch to the gpadmin user.  To test the resource queue, you must run a query in another session to see if it is allowed to run while the other session is holding open the resource queue’s active query slot.  From the second terminal window:  Switch users to gpadmin: [gpadmin@mdw ~]$ **su – gpadmin**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbce4f74.PNG |
|  | Log in to psql with <*your\_name*>:  [gpadmin@mdw ~]$ **psql -U student faa**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbcf81d9.PNG |
|  | Run a query in the second terminal session:  faa=> **SELECT \* FROM test\_table2;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbd24843.PNG  Did the query run? What happened? |
|  | Check the resource queue status:  Open a third terminal session. Log in as **gpadmin.** |
|  | Connect to the faa database as gpadmin. Issue the following command to view the state of the resource queues:  faa=# **SELECT \* FROM pg\_resqueue\_status;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbd394cd.PNG  The **rsqwaiters** column shows the number of statements waiting in a queue. The **rsqholders** column shows the number of queries currently running in a queue. |
|  | Leaving the second terminal session open, return to the first terminal session, the one with the open cursor, and end the transaction: faa=> **END;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbd53cfd.PNG |
|  | Examine the second terminal session. The waiting query should have executed immediately after the transaction in the first session was closed.  Exit psql in all three windows. Close all terminal sessions.  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbd5ac05.PNG |
|  | **Summary**  Administrators can create resource queues for the various types of workloads in their organization. The administrator would then set limits on the resource queue based on his/her estimate of how resource intensive the queries associated with that workload are likely to be.  Database roles (users) are then assigned to the appropriate resource queue. A resource queue can support multiple roles. |

*End of Lab Exercise*

This page intentionally left blank

# Lab Controlling Access

|  |  |
| --- | --- |
| **Purpose:** | In this lab, you design and implement the basic security architecture for your data mart. You will create two new users and use groups to control the level of access that users, who will be members of the group, receive. This method makes it easier to administer privileges by working with groups instead of working with individual user accounts. |
|  | |
| **Tasks:** | Students perform the following task: Implement basic security at the group level. |
|  | |
| **References:** | Module 4 – Defining and Securing the User Database   * Lesson: Roles, Privileges, and Resources |

Lab 10: Task – Implement Basic Security at the Group Level

| **Step** | **Action** |
| --- | --- |
|  | Open a terminal session to the master server using the gpadmin user account, if one is not already available. Connect to the faa database as gpadmin**.**  [gpadmin@mdw ~]$ **psql faa** |
|  | Create one of the user roles that will be used to verify the security implementation:  faa=# **CREATE ROLE batchuser LOGIN;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbd6fd7e.PNG |
|  | Change over to the new role you created, **batchuser**:  faa=# **SET ROLE batchuser;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbd7584a.PNG |
|  | Query the test\_table table:  faa=> **SELECT \* FROM faadata.test\_table;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbd7c233.PNG  Did it succeed? If not, why not? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Change back to gpadmin:  faa=> **SET ROLE gpadmin;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbd8270d.PNG |
|  | Grant permissions to the faadata schema:  faa=# **GRANT USAGE ON SCHEMA faadata TO batchuser;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbd87e51.PNG  While you have been granted USAGE access on the schema, you still do not have access to the existing tables that do not specifically grant access to all. |
|  | Grant access of our test\_table to the group:  faa=# **GRANT select, insert, update, delete ON faadata.test\_table TO batchuser;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbd8dd03.PNG |
|  | Try accessing the faadata**.**test\_table again as batchuser. Change over to the new role:  faa=# **SET ROLE batchuser;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbd93b49.PNG |
|  | Query the test\_table table:  faa=> **SELECT \* FROM faadata.test\_table;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbd994cd.PNG  You should now have SELECT, INSERT, DELETE, and UPDATE access to the test\_table table in the faadata schema. |
|  | Create a group role and assign privileges indirectly. First, change back to the gpadmin role:  faa=> **SET ROLE gpadmin;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbd9f1cb.PNG |
|  | Revoke privilege to the faadata**.**test\_table table as gpadmin:  faa=# **REVOKE SELECT, INSERT, UPDATE, DELETE ON faadata.test\_table from batchuser;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbda4aa4.PNG |
|  | Revoke privileges to the **faadata** schema:  faa=# **REVOKE USAGE ON SCHEMA faadata FROM batchuser;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbdaadf8.PNG |
|  | Create the batch group role that you will assign privileges to:  faa=# **CREATE ROLE batch;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbdb1301.PNG |
|  | Grant permissions to the faadata schema to the batch role you created:  faa=# **GRANT USAGE ON SCHEMA faadata TO batch;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbdb781a.PNG |
|  | Grant access on the test\_table to the group, batch:  faa=# **GRANT SELECT, INSERT, UPDATE, DELETE ON faadata.test\_table TO batch;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbdbd16f.PNG  Members of the group, batch, will now have SELECT, INSERT, UPDATE, and DELETE privileges on the faadata.test\_table table. |
|  | Assign a user role to the newly created group role:  faa=# **GRANT batch TO batchuser;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbded594.PNG |
|  | Create a second user role, batchuser2:  faa=# **CREATE ROLE batchuser2 login;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbdf34f2.PNG |
|  | Assign the batchuser2 user role to the batch group role:  faa=# **GRANT batch TO batchuser2;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbdf8b3c.PNG |
|  | Verify that both user roles have access the faadata.test\_table table. First, change to the batch user role:  faa=# **SET ROLE batchuser;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbe000f8.PNG |
|  | Query the test\_table table:  faa=> **SELECT \* from faadata.test\_table;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbe05a9b.PNG  You should have access to the table because of the role’s relationship to the batch role. |
|  | Change to the batchuser2 role and verify this user has access to the same table:  faa=> **SET ROLE batchuser2;**  faa=> **SELECT \* from faadata.test\_table;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbe0e34a.PNG  This role has also been granted access to the table due to its association with the batch role. |
|  | Tie all users in the batch group into the adhoc resource queue you created earlier:  Change to the gpadmin role:  faa=> **SET ROLE gpadmin;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbe19367.PNG |
|  | Add the batch role to the resource queue, adhoc:  faa=# **ALTER ROLE batch RESOURCE QUEUE adhoc;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLbe22874.PNG  Exit psql.  faa=# **\q** |
|  | **Summary**  You should work with the user community and the administrators to come up with a security architecture that will fit your access requirements.  Table and schema level access is done through the use of roles.  Row level access is best accomplished with a combination of roles and views against the data that has all of the security logic built into the view. While this may inhibit performance for some queries, it will also ensure that your data is secure and only “need to know” access is given to the end users.  Row level security may also be performed by reporting engines. This may suffice if your data security level is low. |

*End of Lab Exercise*

# Lab Table Management

|  |  |
| --- | --- |
| **Purpose:** | The purpose of this lab is to create various types of supported Greenplum Database tables. You will create tables of varying types, load data to these tables, and discover which table type offers better performance under specific conditions. |
|  | |
| **Tasks:** | Students perform the following tasks:   * + Create a temporary table   + Verify the size of two identical tables where one is compressed and the other is not   + Verify the performance on a column-oriented table |
|  | |
| **References:** | Module 5 – Data Loading and Distribution   * Lesson: Implementing Table Storage Models, Compression, and Tablespaces |

Lab 11: Task 1 – Creating Temporary Tables

| **Step** | **Action** |
| --- | --- |
|  | Create the dbstudent database which will be used to support your tables.  [gpadmin@mdw ~]$ **createdb dbstudent**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLe4623.PNG |
|  | Connect to the dbstudent database:  [gpadmin@mdw ~]$ **psql dbstudent**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLf7f5c.PNG |
|  | Create a new schema called studentdata.  dbstudent=# **create schema studentdata;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML19f076.PNG |
|  | Change the search\_path parameter on the database dbstudentso that the new studentdata schema is the default schema.  dbstudent=# **alter database dbstudent set search\_path to studentdata,public,pg\_catalog;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML21b615.PNG  Reconnect to the database to re-read the search\_path settings.  dbstudent=# **\c dbstudent** |
|  | Create a temporary table called dimairline\_t using the following SQL syntax:  dbstudent=# **CREATE TEMPORARY TABLE DimAirline\_T(**  **AirlineID Smallint,**  **AirlineName Character varying(95))**  **WITH (OIDS=FALSE)**  **DISTRIBUTED BY (AirlineID);**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML22c523.PNG |
|  | List all the tables in the dbstudent database.  dbstudent=# **\dt**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML238e3b.PNG  Note that the table is listed in a temporary schema. The schema name may vary from what is displayed in this step. Any other temporary tables created in this session will be saved to the same temporary schema. |
|  | Reconnect to the dbstudent database.  dbstudent=# **\c dbstudent** |
|  | List all the tables in the dbstudentdatabase.  dbstudent=# **\dt**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML28bf39.PNG  The temporary schema that was created when you created the temporary table is automatically dropped when the session ends. Any tables in the schema are also dropped. Reconnecting to the database creates another temporary schema which will be used should you create temporary tables in the new session. |
|  | **Summary**  Temporary tables exist in a special schema, so a schema name may not be given when creating a temporary table. When creating temporary tables, the table name must be unique to the session.  Different sessions can use the same name as the schema name will differ from session to session.  Temporary tables are automatically dropped at the end of the session or optionally, at the end of a transaction with the ON COMMIT clause. This will occur when dropping the session or reconnecting to it, something that occurs often with business intelligence applications and reporting tools.  Temporary tables are a good way of handling complex and intensive SQL statements used for generating reports. They can be used to reduce performance impacts for queries that may have a tendency to generate computational skew. |

Lab 11: Task 2 – Creating Compressed Tables

| **Step** | **Action** |
| --- | --- |
|  | If not already connected to the database, connect to the dbstudent database.  [gpadmin@mdw ~]$ **psql dbstudent**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLf7f5c.PNG |
|  | In this task, you will create two tables: a compressed and a regular heap table. These two tables both have an equal number of fields and records.  Create a regular heap table called dimairline as shown below:  dbstudent=# **CREATE TABLE DimAirline(**  **AirlineID Smallint,**  **AirlineName Character varying(95))**  **WITH (OIDS=FALSE)**  **DISTRIBUTED BY (AirlineID);**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML58f171.PNG |
|  | Create a compressed table called x\_airline as shown below:  dbstudent=# **create table x\_airline (like dimairline)**  **with (appendonly=true,**  **oids=false,**  **compresstype=zlib,**  **compresslevel=5)**  **distributed by (airlineid);**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML594f1a.PNG |
|  | Populate the heap table as shown.  dbstudent=# **copy dimairline from '/rawdata/FAAData/DimAIRLINES.csv'**  **with delimiter ',' CSV HEADER QUOTE '"';**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML59f9f9.PNG |
|  | Populate the compressed table with records from dimairline table as shown.  dbstudent=# **insert into x\_airline select \* from dimairline;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5a60c6.PNG |
|  | Verify the size of both tables by viewing the sotdsize column of the gp\_toolkit.gp\_size\_of\_table\_disk table. This table stores information on all tables within the database. The size column, sotdsize, is displayed in bytes.  dbstudent=# **select sotdsize from gp\_toolkit.gp\_size\_of\_table\_disk**  **where sotdtablename = 'dimairline';**  dbstudent=# **select sotdsize from gp\_toolkit.gp\_size\_of\_table\_disk**  **where sotdtablename = 'x\_airline';**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML5c738b.PNG  Note that the listed size of the tables differ greatly. The compressed AO table, x\_airline, is less than 25% of the size of the heap table, dimairline. |
|  | Exit the database once you have completed the steps.  dbstudent=# **\q** |
|  | **Summary**  Both tables have the same number of fields and records. However, the compressed table is considerably smaller in terms of the amount of its disk size than the uncompressed table. Tables that use compression must be an append-only table. You can only add new rows to an append-only table. You cannot update or delete rows from an append-only table. This allows for more compact storage on disk. It saves 20 bytes per row.  Choosing greater levels of compression result in more compact storage. Increasing compression requires more CPU cycles to access the data when required, either on reads or writes. For data that is not often accessed, it may be applicable to apply strong compression to the table, or partition. This ensures that older data that is not often accessed do not consume large amounts of storage and should balance well against how often you need to access them. |

Lab 11: Task 3 – Creating a Column-Wise Storage Table

| **Step** | **Action** |
| --- | --- |
|  | Create two tables in the dbstudentdatabase. The tables will be called factontimeperformance and c\_factontimeperformance**.** Thefactontimeperformancetable is a regular heap table and c\_factontimeperformanceis acolumn-wise storage oriented table. Run the script below to create the heap table.  [gpadmin@mdw ~]$ **psql -f \ /rawdata/FAAData/CreateDbstudent\_Performance.sql dbstudent**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML67fcb7.PNG |
|  | The heap tablefactontimeperformancehas been created.  Connect to the dbstudent database and create the compressed table, c\_factontimeperformance,basedfactontimeperformanceas shown.    [gpadmin@mdw ~]$ **psql dbstudent**  dbstudent=# **create table c\_factontimeperformance   (like factontimeperformance)**  **with (appendonly=true, orientation=column)**  **distributed randomly;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML686421.PNG |
|  | Use the \i command to execute scripts that will populate the tables you have just created. It may take a few minutes to load all of the data.  dbstudent=# **\i /rawdata/FAAData/load\_facton2008.sql**  dbstudent=# \i **/rawdata/FAAData/load\_c\_facton2008.sql**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML72c2e3.PNG  After running these scripts, each table should contain 7 million records. The data loaded into both tables represent the entire year of 2008 for FAA flight data. |
|  | Verify you have the same number of records on both tables.  dbstudent=# **select** **count(\*) from factontimeperformance;**  dbstudent=# **select count(\*) from c\_factontimeperformance;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML758863.PNG |
|  | Calculate how many times Southwest Airlines flew to Chicago in the year 2008 running the query below. Run the query on both tables: factontimeperformance and c\_factontimeperformance. You should see the same results for both tables.  dbstudent=# **select count(\*) from factontimeperformance where**  **year = 2008 and**  **airlineid = 19393 and**  **destcityname = 'Chicago, IL';**  dbstudent=# **select count(\*) from c\_factontimeperformance where**  **year = 2008 and**  **airlineid = 19393 and**  **destcityname = 'Chicago, IL';**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML767df7.PNG |
|  | To verify which query is faster, analyze the queries using explain analyze command. This command will be taught in more detail in later labs. For now, pay close attention to the **Total runtime** line.  Run the command for the factontimeperformance table, which is a row-oriented heap table.  dbstudent=# **EXPLAIN ANALYZE select count(\*) from factontimeperformance where year = 2008 and  airlineid = 19393 and  destcityname = 'Chicago, IL';**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML7b88ff.PNG  In this example, the query took 9751.207ms to complete. Your time may vary from what is displayed here. Compare these results against the append-only column-oriented table. |
|  | Execute the same command for the c\_factontimeperformance table, which is an append-only column oriented table.  dbstudent=# **EXPLAIN ANALYZE select count(\*) from c\_factontimeperformance where year = 2008 and  airlineid = 19393 and  destcityname = 'Chicago, IL';**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML7d5ee3.PNG  In this example, the query took 2146.133 ms to complete. Your time may vary from what is displayed here. You should note that the completion time on the column-oriented table is significantly less than the heap table. As there are few columns thar are being accessed, column-oriented tables offer an advantage in this query. |
|  | Exit the database. |
|  | **Summary**  For most general purpose or mixed workloads, row-oriented storage offers the best combination of flexibility and performance. However, there are certain specific use cases where a column-oriented storage model provides more efficient I/O and storage.  Column-oriented tables offer strong performance gains over row-oriented tables when the table is being used mostly for reads and there are few columns being selected against or aggregated over. For data that is not updated often or is limited to a single column, column-oriented tables may offer a distinct advantage for gathering data from the table over row-oriented tables. A negative impact can be seen if you need to select a majority of the columns on a column-oriented table. Understanding the data and how it will be used will help you choose the appropriate storage model for the table. |

*End of Lab Exercise*

This page intentionally left blank

# Lab Data Loading

|  |  |
| --- | --- |
| **Purpose:** | The purpose of this lab is to learn the different techniques for loading data into a Greenplum system.  You will start by using the COPY command. The COPY command is well suited for importing small amounts of data in text or CSV format.  The second technique you will learn for loading data is creating external tables and using an insert statement from the external table into an internal Greenplum table.  The third technique that you will learn for loading data is using the gpfdist utility. |
|  | |
| **Tasks:** | Students perform the following tasks:   * + Create a dimension table and load data using COPY   + Create a dimension table and load data using external tables   + Create a fact table and load data using gpfdist   + Create a writeable external table to populate a different database   + Load data from a compressed file with gpfdist |
|  | |
| **References:** | Module 5 – Data Loading and Distribution   * Lesson: Data Loading |

Lab 12: Task 1 – Create a Dimension Table and Load Data Using COPY

| **Step** | **Action** |
| --- | --- |
|  | Connect to the faa database as the gpadmin user:  [gpadmin@mdw ~]$ **psql faa**  C:\Users\cantot\AppData\Local\Temp\SNAGHTMLb1dd09.PNG |
|  | In a separate terminal session where you have a psql session started, create the first Dimension table DimAirline:  faa=# **Create table DimAirline(**  **AirlineID Smallint,**  **AirlineName Character varying(95))**  **WITH (OIDS=FALSE)**  **DISTRIBUTED BY (AirlineID);**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1274983.PNG  Note: You can also copy and paste this command from the script DDLScript.sql located in /rawdata/FAAData. |
|  | From the psql session, use the COPY command to populate the DimAirline table using the data from the CSV file:  faa=# **COPY DimAirline FROM '/rawdata/FAAData/DimAIRLINES.csv'**  **WITH DELIMITER ',' CSV HEADER QUOTE '"'**  **SEGMENT REJECT LIMIT 10 ROWS;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1283900.PNG  Use single quotes to refer to the file in the COPY command. The delimiter used in this file is the comma. The data file is a CSV file with a header. Quoted items are surrounded with double quotes.  In this example, a maximum of ten (10) row failures are allowed before the copy fails.  Note: You can also copy and paste the above command from the script CopyData.sql located on /rawdata/FAAData**.** |
|  | Verify that the table is now populated with data:  faa=# **SELECT \* FROM DimAirline;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML128f913.PNG  Hit q to quit scrolling through the data or use the spacebar to scroll through all of the data. |
|  | **Summary**  The COPY command can be used to load data from a file (or from standard input) into a table. COPY can be run as a single command in non-interactive mode or in an interactive psql session. COPY does not load data in parallel, so it is better suited for loading smaller amounts of data. If loading large amounts of data, external tables offer better load performance.  If your data load file contains mixed data, you can use COPY to quickly load the data into a temporary table and then insert it from there through the parent table. |

Lab 12: Task 2 – Create a Dimension Table and Load Data Using External Tables

| **Step** | **Action** |
| --- | --- |
|  | Define an external table called public.airport\_external that points to a CSV file located on another server. The file can be accessed as file://sdw1/loaddata/DimAIRPORTS.csv. To create the table, use the CREATE EXTERNAL TABLE command:  faa=# **CREATE EXTERNAL TABLE public.airport\_external**  **(AirportID Character(3),**  **AirportDescription Character varying(85))**  **LOCATION ('file://sdw1/loaddata/DimAIRPORTS.csv')**  **FORMAT 'CSV' (HEADER)**  **LOG ERRORS INTO public.airport\_err**  **SEGMENT REJECT LIMIT 10 ROWS;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML132123e.PNG  Note:When using the file:// protocol, the external data file(s) must reside on a segment host in a location accessible by the Greenplum super user (gpadmin). |
|  | Verify that you are able to access data from the external table by counting the number of records:  faa=# **SELECT COUNT(\*) FROM public.airport\_external;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1338cd9.PNG  If you receive an error, verify the syntax you used to create the external table, the host name for the file, and the file name. Verify that the file is in the location you specified in the external table syntax. If necessary, drop the external table using DROP EXTERNAL TABLE public.airport\_external and recreate the external table again. |
|  | Create a standard table to copy the external data into:  faa=# **CREATE TABLE faadata.DimAirport(**  **AirportID Character(3),**  **AirportDescription Character varying(85))**  **WITH (OIDS=FALSE)  DISTRIBUTED BY (AirportID);**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML13402e3.PNG  **Note:** You can also copy and paste this command from the DDLScript.sql script. |
|  | Insert the data into the DimAirport from the external table:  faa=# **INSERT INTO faadata.DimAirport**  **SELECT \* FROM public.airport\_external;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML134c769.PNG |
|  | Verify that the dimension table now contains data:  faa=# **SELECT \* FROM faadata.DimAirport LIMIT 10;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1352c34.PNG  Note: The LIMIT clause returns the specified number of rows to the user, reducing the amount of memory required by the client to handle the number of rows returned by Greenplum. This can be helpful if your query returns a large number of rows but you wish to only view a subset of these rows. |
|  | Exit the database before proceeding. |
|  | **Summary**  Both CREATE EXTERNAL TABLE and COPY operations can be run using the single row error isolation feature. This feature allows you to load good rows while filtering out error rows. Error rows can be logged in to an error table for further examination. External tables offer additional flexibility over COPY because you can use regular SQL commands to select and move the data prior to inserting it into its final destination tables. This is useful for ETL processing common in data warehousing applications.  It is good practice to automatically run ANALYZE after any data load so that the query planner has the most up-to-date statistics. If you had any errors during your data loads, it is also a good idea to run VACUUM to reclaim any wasted space. |

Lab 12: Task 3 – Create a Fact Table and Load Data using gpfdist

| **Step** | **Action** |
| --- | --- |
|  | If not already connected to the master server, open a terminal session to the master server and connect as gpadmin. |
|  | Start a gpfdist session.  [gpadmin@mdw ~]$ **gpfdist -d /rawdata/FAAData -p 8081 \ -l /home/gpadmin/log &**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML136d9a2.PNG  This starts a parallel file distribution session on port 8081 for files in the directory, /rawdata/FAAData. These files can now be read into the database from an external table. |
|  | Verify that gpfdist is running:  [gpadmin@mdw ~]$ **pgrep gpfdist**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML13765d9.PNG |
|  | Examine the contents of the /rawdata/FAAData/CreateExt-OnTimePerformance.sql file and go to the end of it. You will not be making any changes to this file. |
|  | This file will create an external table as shown below.  CREATE EXTERNAL TABLE public.FactOnTimePerformance\_external (…  ) LOCATION ('gpfdist://mdw:8081/On\_Time\_On\_Time\_Performance\_2008\_1.csv',  'gpfdist://mdw:8081/On\_Time\_On\_Time\_Performance\_2008\_2.csv') FORMAT 'CSV' (HEADER DELIMITER ',') LOG ERRORS INTO public.fact\_err SEGMENT REJECT LIMIT 10 ROWS; |
|  | Execute the script as shown.  [gpadmin@mdw ~]$ **psql \ -f /rawdata/FAAData/CreateExt-OnTimePerformance.sql -d faa**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML13b58df.PNG  The output highlights that the fact\_err table did not previously exist. One will be created to capture any failures. The last portion of the output highlights that a header is expected of all files loaded through this external table. |
|  | Connect to the faa database.  [gpadmin@mdw ~]$ **psql faa**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML13fde05.PNG |
|  | Confirm that the contents of the files can now be accessed through the external table created:  faa=# **SELECT \* from public.factontimeperformance\_external limit 10;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML140a27c.PNG |
|  | Create a regular heap table based on the external table and populate it with the data from the external table:  faa=# **create table faadata.factontimeperformance**  **As**  **Select \* from public.factontimeperformance\_external**  **Distributed randomly;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML14214fc.PNG |
|  | Confirm that the faadata.factontimeperformance table has been created and populated:  faa=# **SELECT \* from faadata.factontimeperformance limit 10;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1431a69.PNG  The LIMIT 10 clause lists only 10 of the possible rows. |
|  | Execute the /rawdata/FAAData/DropTables.sql script to drop the tables created. This script will drop the dimairline, dimairport, and factontimeperformance tables.  faa=# **\i /rawdata/FAAData/DropTables.sql**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML14768f5.PNG |
|  | The next two steps will create and populate the tables that will be used in future labs.  Create the tables by executing the script DDLScript.sql**.**  faa=# **\i /rawdata/FAAData/DDLScript.sql**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML14b6473.PNG |
|  | Load data for the tables you just created in the faa schema by executing the script, CopyData.sql. The data is approximately a total of 20M rows and may take 5-10 minutes to populate.  faa=# **\i /rawdata/FAAData/CopyData.sql**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1513910.PNG |
|  | Exit the database. |
|  | **Summary**  The parallel file distribution program can be used in conjunction with gpload or other ETL tools to take advantage of the parallel load. It starts a light-weight session on the port specified and can be used to load multiple files from the same directory.  You can start multiple gpfdist processes to maximize the transfer of data. |

Lab 12: Task 4 – Create a Writeable External Table to Load Data into a Different Database using gpfdist

| **Step** | **Action** |
| --- | --- |
|  | If not already connected to the master server, open a terminal session to the master server and connect as gpadmin. |
|  | Create a directory called data in your home directory that you will use in this lab. This directory will be used by writable tables to create files with data pushed from the database.  [gpadmin@mdw ~]$ **mkdir data**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML155a337.PNG |
|  | Start a gpfdist session that will access files in the ~/data directory you created. Start this gpfdist session on port 8082. You must have write permission on the directory you are using for this session.  [gpadmin@mdw ~]$ **gpfdist -d ~/data -p 8082 \**  **-l /home/gpadmin/log &**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1572217.PNG  This starts a parallel file distribution session on port 8082 for files in the directory, /home/gpadmin/data. |
|  | Verify that gpfdist is running for the newly created session:  [gpadmin@mdw ~]$ **ps -ef | grep gpfdist**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15a1b63.PNG  You may see a second gpfdist session running for port 8081 created earlier in the lab. |
|  | Connect to the faa database as the gpadmin user.  [gpadmin@mdw ~]$ **psql faa**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15e944a.PNG |
|  | Create a writable external called XTABLE\_EXT0. The table will be a copy of the structure of the dimairporttable. The table should point to the wtable file through the gpfdist session running on port 8082.  faa=# **CREATE WRITABLE EXTERNAL TABLE XTABLE\_EXT0 (LIKE DIMAIRPORT)**  **LOCATION ('gpfdist://mdw:8082/wtable')**  **FORMAT 'TEXT' (DELIMITER '|');**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15f1c6c.PNG  The gpfdist process is pointing to /home/gpadmin/data on the master server, mdw. This writeable table is therefore pointing to the file, /home/gpadmin/data/wtable. Data with multiple columns will be separated with the pipe (|) symbol. |
|  | Populate the wtable file with content from the dimairport table. This is known as unloading data. It does not, however, remove data from the dimairport table.  faa=# **insert into xtable\_ext0 select \* from dimairport;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15f7fff.PNG |
|  | Verify that the contents of the file have been populated with the data contained in the dimairport table. Use the escape shell (!) to execute the more command for the /home/gpadmin/data/wtable file.  faa=# \! **more /home/gpadmin/data/wtable**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML16b1d95.PNG  Hit **q** to exit. |
|  | Create a new database called dbbackup.  faa=# **create database dbbackup;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML16016c1.PNG |
|  | Connect to the dbbackup database.  faa=# **\c dbbackup**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1607a73.PNG |
|  | Create a new schema for the dbbackupdatabase called backupdata**.**  dbbackup=# **create schema backupdata;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1615b40.PNG |
|  | Change the search\_ path parameter on the dbbackup database so that the backupdata schema is the default schema.  dbbackup=# **alter database dbbackup set search\_path to backupdata,public,pg\_catalog;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML161b60c.PNG |
|  | Reconnect to the dbbackup database. This step is necessary for reloading the new search\_pathparameter.  dbbackup=# **\c dbbackup**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML16207f3.PNG |
|  | Create dimairporttable in the dbbackupdatabase using the following syntax:  dbbackup=# **create table dimairport (**  **airportid character(3),**  **airportdescription character varying(85))**  **with (oids=false);**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML162604f.PNG |
|  | Populate the dimairport table with the content from the /home/gpadmin/data/wtable file.  dbbackup=# **copy dimairport from '/home/gpadmin/data/wtable'  with delimiter '|';**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML167266b.PNG |
|  | **Summary**  Writable external tables are used to select rows from database tables and output the rows to files, named pipes, or to other executable programs. You could unload data from a Greenplum Database table and send that data to a text file. Once the text file is created, you can use it to populate another Greenplum Database table. This technique, shown in the exercise you just completed, is an example of migrating data from one database to another. If necessary, you can perform transforms on the data when inserting it into the database.  Writable external tables only allow INSERT operations. |

Lab 12: Task 5 – Loading a Compressed File using gpfdist

| **Step** | **Action** |
| --- | --- |
|  | Connect to the faa database as gpadmin.  dbbackup=# **\c faa** |
|  | Create an externaltable called public.gztable based on the DDL of the factontimeperformance table. Populate public.gztable with the contents from a compressed file without first decompressing it.  faa=# **create external table public.gztable   (like factontimeperformance)**  **location  ('gpfdist://mdw:8081//On\_Time\_On\_Time\_Performance\_2008\_1.csv.gz')**  **format 'csv' (header delimiter ',');**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML18557e7.PNG |
|  | Verify the external table has been created listing external tables with the \dx command.  faa=#\dx  C:\Users\cantot\AppData\Local\Temp\SNAGHTML185ac7c.PNG |
|  | To make the output easier to see, complete the following to change the output mode in PSQL:   * Issue the following command to eliminate the header from the output: faa=# **\t on** * Issue the following command to change to unaligned output. The default separator for unaligned output is the pipe (|) symbol: faa=# **\a** * Issue the following command to change the field separator to a comma (,): faa=# **\f ,**   C:\Users\cantot\AppData\Local\Temp\SNAGHTML18dc04a.PNG |
|  | List two records from this external table as shown.  faa=# **select \* from gztable limit 2;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML18e5c2b.PNG |
|  | Reset your output by issuing the following PSQL meta commands:  faa=# **\t off**  faa=# **\a**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1927ed7.PNG |
|  | Create a new table called faadata.performance and populate it using the data from the external table,public.gztable.  faa=# **create table faadata.performance as**  **select \* from public.gztable**  **distributed randomly;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML19483f6.PNG |
|  | Exit the database. |
|  | **Summary**  Data in files compressed with gzip or bzip2, can be loaded into Greenplum Database tables without uncompressing the files. This saves storage space on the host serving these files. |

*End of Lab Exercise*

# Lab Table Partitioning

|  |  |
| --- | --- |
| **Purpose:** | The purpose of this lab is to learn about Greenplum’s partitioning capabilities. Partitioning is typically used on large fact tables to improve performance and manageability. Greenplum supports single level range and list partitioning. You will learn how to create and manage partitioned tables. |
|  | |
| **Tasks:** | Students perform the following task: create and manage table partitions. |
|  | |
| **References:** | Module 5 – Data Loading and Distribution   * Lesson: Table Partitioning |

Lab 13: Task – Create and Manage Table Partitions

| **Step** | **Action** |
| --- | --- |
|  | Connect to the faa database as the gpadmin user, if not already connected:  [gpadmin@mdw ~]$ **psql faa** |
|  | Make sure that the partitioned tables feature is turned on. This allows the query planner to selectively scan table partitions at runtime. Confirm that constraint\_exclusion has a value of **on**:  faa=# **SHOW constraint\_exclusion;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML196173f.PNG |
|  | Create a partitioned version of the fact table. Each month will be stored in a separate partition. The partitioned table will be based on the faadata.FactOnTimePerformance table created in an earlier lab.  faa=# **CREATE TABLE faadata.factontimeperformance\_parted**  **(LIKE faadata.FactOnTimePerformance)**  **DISTRIBUTED BY (AirlineID)**  **PARTITION BY RANGE ( Year ) (**  **START (2008)**  **END (2011)**  **EVERY (1)**  **);**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML198c098.PNG |
|  | List the tables:  faa=# **\dt**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML199af98.PNG  You should note that the faadata.factontimeperformance\_parted table was created along with 3 child tables. |
|  | Check the definition of one of the child tables:  faa=# **\d+ faadata.factontimeperformance\_parted\_1\_prt\_1**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML19e3367.PNG  C:\Users\cantot\AppData\Local\Temp\SNAGHTML19f2534.PNG  You should see that the child table has the same structure as the parent table, but also includes a check constraint for the year. |
|  | Load data into the partitioned table to verify that the partition scheme is working:  faa=# **COPY faadata.FactOnTimePerformance\_parted FROM '/rawdata/FAAData/On\_Time\_On\_Time\_Performance\_2008\_1.csv'**  **WITH DELIMITER ',' CSV HEADER QUOTE '"';**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a28df5.PNG  **Note:** There is adouble quote in the middle of two single quotes at the end of the line. The filename specified in the command is case sensitive.  **Note:** You are loading data into the parent table, not the child table. If the partitioning is working correctly, Greenplum will place the data into the appropriate partition when data is inserted into the parent table. |
|  | Verify that the data was copied into the correct partition:  faa=# **SELECT COUNT(\*)from faadata.FactOnTimePerformance\_parted;**  faa=# **SELECT COUNT(\*)from faadata.FactOnTimePerformance\_parted\_1\_prt\_1;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a309d8.PNG  **Note:** All of the records were placed into the first child partition table. If you query against any of the other child tables, you will find that they do not contain any rows. |
|  | Load the January 2011 data into the same table:  faa=# **COPY faadata.FactOnTimePerformance\_parted FROM '/rawdata/FAAData/On\_Time\_On\_Time\_Performance\_2011\_1.csv' WITH DELIMITER ',' CSV HEADER QUOTE '"';**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a92850.PNG  This operation should fail since the partition for 2011 has not been created yet. There is also no default partition to capture any data that does not fall into the range of any defined partitions. |
|  | Create a new partition for the 2011 data:  **ALTER TABLE faadata.factontimeperformance\_parted   add partition Y2011  START (smallint '2011')  END (smallint '2012');**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a9ea95.PNG |
|  | Load the January 2011 data again:  faa=# **COPY faadata.FactOnTimePerformance\_parted FROM '/rawdata/FAAData/On\_Time\_On\_Time\_Performance\_2011\_1.csv'**  **WITH DELIMITER ',' CSV HEADER QUOTE '"';**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1aa92e4.PNG  Note: The operation should now succeed. |
|  | The table and partition names can be long and non-intuitive. Rename the table to factotperf:  faa=# **ALTER TABLE factontimeperformance\_parted RENAME TO factotperf;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1ab3f78.PNG |
|  | Verify that the parent table and the child tables have been renamed:  faa=# **\dt**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1ac2e1a.PNG |
|  | Rename the 2008 partition so that it has a more intuitive name:  faa=# **ALTER TABLE factotperf RENAME PARTITION for (2008) to Y2008;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1accd74.PNG |
|  | Verify that the partition has been renamed:  faa=# **\dt** C:\Users\cantot\AppData\Local\Temp\SNAGHTML1adb302.PNG |
|  | You may need to remove a partition after a period of time to age out data. Drop the 2008 partition with the following command:  faa=# **ALTER TABLE factotperf DROP PARTITION FOR (2008);**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1ae1887.PNG |
|  | Exchanging a partition allows you to bring data from a table into a partition. Create a table named fact\_temp that has the same structure as factotperf. This table will contain the data that will eventually be placed into the partitioned table:  faa=# **CREATE TABLE fact\_temp (LIKE factotperf);**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1ae7ea9.PNG |
|  | Remove any existing data from the factotoperf table by truncating the table. This will truncate all partitions within the table. faa=# **TRUNCATE TABLE factotperf;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1aecdd2.PNG |
|  | Copy the January 2011 data into fact\_temp:  faa=# **COPY faadata.fact\_temp FROM '/rawdata/FAAData/On\_Time\_On\_Time\_Performance\_2011\_1.csv'**  **WITH DELIMITER ',' CSV HEADER QUOTE '"';**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1afb237.PNG |
|  | Take the data found in the factotperf table and push it to the Y2011 partition using the EXCHANGE clause on the partition. This will also move the data that was in the partition to the fact\_temp table:  faa=# **ALTER TABLE factotperf EXCHANGE PARTITION Y2011 WITH TABLE fact\_temp;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1be23ce.PNG |
|  | Verify that the data is now in the partitioned table and that the fact\_temp table has no rows:  faa=# **SELECT COUNT(\*) FROM factotperf;**  faa=# **SELECT COUNT(\*) FROM fact\_temp;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1c02303.PNG |
|  | Sometimes a partition may become too large and it might be worthwhile to split the partition. Create a new partitioned table:  faa=# **CREATE TABLE ORDERS(ordered int, orderDate date)**  **DISTRIBUTED BY (ordered)**  **PARTITION BY RANGE (orderDate)**  **(START ('2008-01-01')**  **END ('2008-12-31')**  **EVERY (INTERVAL '1 month')**  **);**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1c3ae51.PNG |
|  | Split the January partition into two parttions:  faa=# **ALTER TABLE orders**  **SPLIT PARTITION FOR ('2008-01-01')**  **AT ('2008-01-16')**  **INTO (PARTITION jan20081to15, PARTITION jan200816to31);**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1c41750.PNG |
|  | Verify that the new partitions have been created:  faa-# **\dt**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1c4dfad.PNG  Exit the database before proceeding. |
|  | **Summary**  Table partitioning addresses the problem of supporting very large tables, such as fact tables, by allowing you to logically divide them into smaller and more manageable pieces. Partitioning is used to improve performance by scanning only the relevant data needed to satisfy a query. It can also facilitate database loading and maintenance.  In the Greenplum Database, partitioning is a procedure that creates multiple sub-tables (or child tables) from a single large table (or parent table) and setting exclusion constraints on the child tables. Table data resides in the child tables only; no data should be loaded in the parent tables. Note that partitioned tables in the Greenplum Database are also physically distributed across the segment instances just as are non-partitioned tables.  The ALTER TABLE syntax for handling partitions is very flexible. It allows almost as much control over the structure of a table as the CREATE TABLE syntax. With this one command you are given the ability to add partitions, drop partitions, and rename partitions. Remember that if you drop a partition, it will remove the partition and all dependents.  Exchanging a partition is an easy way to facilitate tricky operations that with other databases would require locking out the users or running late at night so as not to interfere with work. By running a load on a table that is not being used, and then quickly exchanging it once the load is finished, you can see more data security and less downtime. The advantages of exchanging a partition do not end there.  The alter table syntax can be used to split a partition if the partition is growing too large. To use this command, you must split your partition on the partitioning key. The value specified will land in the latter of the two partition tables created. |

*End of Lab Exercise*

This page intentionally left blank

# Lab Managing the Greenplum Database

|  |  |
| --- | --- |
| **Purpose:** | In this lab, you will start, stop, and restart the Greenplum Database. You will also execute commands that provide information on the state of Greenplum.  Greenplum clients must be executed from the master instance while logged in as a Greenplum superuser account. |
|  | |
| **Tasks:** | Students perform the following task: Perform system administration tasks on the Greenplum environment. |
|  | |
| **References:** | Module 6: Database Management and Archiving   * Lesson: Managing the Greenplum Database |

Lab 14: Task – Perform System Administration Tasks on the Greenplum Environment

| **Step** | **Action** |
| --- | --- |
|  | If not already connected to mdw, log in as gpadminon your master server and stop the Greenplum database with the following command :  [gpadmin@mdw ~]$ **gpstop**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML152d619e.PNG  When prompted, answer **y** to proceed with shutting down the instances. |
|  | The gpstart command is used to start the Greenplum database defined by the $MASTER\_DATA\_DIRECTORY parameter or as specified with the -d option. To start the Greenplum database in restrictive mode where only the Greenplum superuser can connect, issue the following command in UNIX as gpadmin:  [gpadmin@mdw ~]$ **gpstart -R**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1547d1fd.PNG  When prompted, answer **y** to proceed with starting the instances. You can skip any prompts by including the -a option as a part of the command.  This mode is useful if you need to perform some isolated work in the environment and need to ensure other users cannot connect to the Greenplum Database. |
|  | Verify the state of the cluster to ensure that the database is running:  [gpadmin@mdw ~]$ **gpstate**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML154bb615.PNG |
|  | Verify the student account that you created in earlier labs cannot connect to the database.  [gpadmin@mdw ~]$ **psql -U student faa**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1551b0c7.PNG |
|  | There are times when you want to change parameters of the database by editing either pg\_hba.conf or postgresql.conf, but do not want to shut down the database to make it aware of the changes. The gpstop -u command lets the master re-read the configuration files without shutting down services.  Using vi, edit the pg\_hba.conf file and add the following line at the end:  **host all all 127.0.0.1/32 trust**  The new entry allows all users from 127.0.0.1 to connect to all of the databases within the Greenplum Database cluster.  [gpadmin@mdw ~]$ **vi $MASTER\_DATA\_DIRECTORY**/pg\_hba.conf  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1554df27.PNG |
|  | Read the new configuration into the database by issuing the following command:  [gpadmin@mdw ~]$ **gpstop -u**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML155622ac.PNG  This will reload all the configuration files without shutting the database down. |
|  | Log in to PSQL and check for skew on the main fact table, factontimeperformance:  [gpadmin@mdw ~]$ **psql faa** |
|  | faa=# **SELECT gp\_segment\_id, count(\*) FROM factontimeperformance**  **GROUP BY gp\_segment\_id;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1558abf7.PNG |
|  | Check for processing skew:  faa=# **SELECT gp\_segment\_id, count(\*) FROM factontimeperformance WHERE cancelled='false' GROUP BY gp\_segment\_id;** C:\Users\cantot\AppData\Local\Temp\SNAGHTML15599f98.PNG |
|  | Exit the database.  faa=# **\q** |
|  | There may be times when you need to perform administrative duties on the segment hosts from the master. You can use the gpssh command from UNIX as gpadmin. The /rawdata/Binaries/hostfile\_exkeys file contains the names of all hosts in this cluster. You can use this file to query the hosts.  [gpadmin@mdw ~]$ **gpssh -f /rawdata/Binaries/hostfile\_exkeys df**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML155e01e3.PNG |
|  | After a database has been running for a long time, its catalog can become bloated, especially if there are a large number of UPDATEs and DELETEs to the database environment. A bloated system requires regular maintenance consisting of scheduled VACUUMs. These systems may periodically require a full VACUUM ANALYZE to clean up space.  It is safer to run a regular VACUUM. A VACUUM FULL can slow down the performance of the system dramatically. It should be executed when in maintenance mode. To run a vacuum on only one catalog table, for example pg\_class, execute the following command:  [gpadmin@mdw ~]$ **psql faa -c 'vacuum analyze pg\_catalog.pg\_class;'**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML156c0222.PNG  The pg\_class table keeps the object names in the database. If there are a lot of updates to the table, such as with creating and dropping objects, the table can become bloated and negatively impact the system.  **Note:** You do not directly make changes to this table. This table is affected by updates in your environment, such as when you create or delete a table. |
|  | To run a VACUUM ANALYZE on all catalog tables create the following script and execute it.   1. Using vi, create a new file called **gp\_vacuum\_analyze** with the following content:   **#!/bin/bash DBNAME="faa" VCOMMAND="VACUUM ANALYZE "**  **psql -tc "select '$VCOMMAND' || ' pg\_catalog.' || relname || ';' from pg\_class a,pg\_namespace b where a.relnamespace=b.oid and b.nspname= 'pg\_catalog' and a.relkind='r'" $DBNAME | psql -a $DBNAME**   1. Save the file and exit vi.   C:\Users\cantot\AppData\Local\Temp\SNAGHTML156e1b6c.PNG   1. Change the permissions on the file so that it is executable: **chmod 755 gp\_vacuum\_analyze** 2. Execute the script by typing the following: **./gp\_vacuum\_analyze** |
|  | C:\Users\cantot\AppData\Local\Temp\SNAGHTML156ef028.PNG  **Note:** Recall that it is recommended that you execute VACUUM on a regular schedule and ANALYZE separately on its own schedule. |
|  | Open a PSQL session and log in as **gpadmin** to the **faa** database.  [gpadmin@mdw ~]$ **psql faa** |
|  | Check the size of the database using the following command:  faa=# **SELECT \* from gp\_toolkit.gp\_size\_of\_database;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML157426f1.PNG |
|  | Retrieve the size of each relational table in user schemas of the user database.  faa=# **SELECT \* from gp\_toolkit.gp\_size\_of\_table\_disk;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15749b36.PNG |
|  | Retrieve the size of all user schemas in the user database.  faa=# **SELECT \* from gp\_toolkit.gp\_size\_of\_schema\_disk;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1578f882.PNG |
|  | Retrieve the size of all user indexes in the user database.  faa=# **SELECT \* from gp\_toolkit.gp\_size\_of\_index;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML157962e7.PNG |
|  | Retrieve the cumulative size of all user indexes in a table, list this total size for each table in the user database.  faa=# **SELECT \* from gp\_toolkit.gp\_size\_of\_all\_table\_indexes;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1579c4e4.PNG |
|  | Retrieve the uncompressed table size.  faa=# **SELECT \* from gp\_toolkit.gp\_size\_of\_table\_uncompressed;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML157a2201.PNG |
|  | Retrieve the total free disk space in Kbytes for each segment server and the file systems included as part of the Greenplum Database.  faa=# **SELECT \* from gp\_toolkit.gp\_disk\_free;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML157b7030.PNG |
|  | Exit the database. |
|  | Restart the database in regular mode so that all users can connect to it. The -r option is used to restart the database.  [gpadmin@mdw ~]$ **gpstop -ar**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1619c610.PNG |
|  | **Summary**  Checking table skew after loading is one of the most important ways to validate the efficiency of your distribution keys. Check the skew often on large tables that are frequently loaded, as data demographics may have changed. This is important on all Greenplum systems regardless of size. On smaller systems, there are fewer segments involved so the distribution is not over as many units of parallelism. On larger systems the hashing algorithm, particularly on non-unique distribution keys, may hash many more rows to a single segment.  Check the size of databases and regularly VACUUM the system to reduce the chances of bloat. |

*End of Lab Exercise*

This page intentionally left blank

# Lab Backups and Restores

|  |  |
| --- | --- |
| **Purpose:** | In this lab, you will schedule a parallel backup operation with the gpcrondump command to dump the faa database. This will create a backup file of the master instance and each active segment instance in the Greenplum Database system.  By default, the dump files are created in the data directory of their respective segment instance or master instance. In this exercise, you will redirect the dump files to one location to make it easier to collect and analyze them.  You will then restore the database using the gprestoredb command. |
|  | |
| **Tasks:** | Students perform the following task:   * Create and retrieve backups * Perform incremental backups and restores * Recover from a failed master |
|  | |
| **References:** | Module 6: Database Management and Archiving   * Lesson: Backups and Restores |

Lab 15: Task 1 – Backups and Restores

| **Step** | **Action** |
| --- | --- |
|  | To create backups you will need a place to keep them. Sometimes it is a network share to a backup devise. Normally all segments will need to have this space. You can perform backups as parallel or non-parallel backups using the following:   * gpcrondump for automatic scheduled parallel backups. * pg\_dump non parallel backup that has to go through the master. (Not recommended because of slow performance) pg\_dump and pg\_restore is available for compatibility with standard postgres databases. |
|  | Start an ad-hoc backup from the UNIX prompt as gpadmin:  [gpadmin@mdw ~]$ **gpcrondump faa**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15cdcafe.PNG  You will be prompted to continue. Type y and press Enter to proceed. |
|  | C:\Users\cantot\AppData\Local\Temp\SNAGHTML15cf258c.PNG |
|  | The backup files will be in the data/db\_dumps directory on the master and all primary segments.  [gpadmin@mdw ~]$ **ls -aR $MASTER\_DATA\_DIRECTORY/db\_dumps**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15de4700.PNG  All backup files are stored in a timestamped directory for the day that the backup was started. In this example, the directory, 20150330, is created when the first backup is executed on March 30th, 2015. Any subsequent backups performed on that day are stored to the same directory. |
|  | Verify the corresponding backup files exist for the segments.  [gpadmin@mdw ~]$ **gpssh -h sdw1 -h sdw2 \ ls -aR /data/primary/gpseg\*/db\_dumps**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15e5a298.PNG |
|  | Over the next few steps, you will configure the environment so that gpcrondump is automated through cron.  The crontab utility determines whether or not a user has appropriate permission to run a program at a particular point in time by checking the file /etc/cron.allow. A user must be explicitly included to this file to be able to use the crontab.  As root, add the gpadmin account to the /etc/cron.allow file in a line by itself. In this example, the cat command is used to add a line to the end of the file. If the file does not exist, it will be created. If it does exist, the double greater than symbols (>>) lets you append to the file.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15e9056f.PNG  To finish modifying the file with the cat command, hit CTRL-D. |
|  | Exit from the root account, back to the gpadmin account.  [root@mdw ~]# **exit**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15ec41b4.PNG |
|  | Ensure that the permissions of .bash\_profile file for the gpadmin user include the execute permission so that cron can properly access and execute the login script:  [gpadmin@mdw ~]$ **chmod +x /home/gpadmin/.bash\_profile**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15ee3268.PNG |
|  | Using vi, add the following lines to the file /home/gpadmin/cronbackup.sh:  **source /home/gpadmin/.bash\_profile** **gpcrondump -x faa -c -g -G -a -q >> /tmp/gpcrondump.log**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15f09899.PNG |
|  | Change the permissions on **cronbackup.sh** to **755** so that it is readable and executable by cron.  [gpadmin@mdw ~]$ **chmod 755 cronbackup.sh**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15f10acc.PNG |
|  | Update the **EDITOR** environment variable to **vi** and export it. When modifying your cron jobs, the editor defined by the EDITOR variable will be used. If you are more familiar with and prefer to use Emacs, you can replace vi shown here with emacs:  [gpadmin@mdw ~]$ **export EDITOR=vi** C:\Users\cantot\AppData\Local\Temp\SNAGHTML15f22944.PNG |
|  | You will set cron to execute the script you created, cronbackup.sh, five (5) minutes from the time you record in this step. Use the date command to obtain the current time:  [gpadmin@mdw ~]$ **date**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15f351f9.PNG |
|  | Edit the crontab and add a line to execute the script you created, cronbackup.sh, five minutes from the time you recorded earlier:  [gpadmin@mdw ~]$ **crontab -e**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15f4c074.PNG  The syntax below is used to show how each crontab line is defined:  **17 6 \* \* \* /home/gpadmin/cronbackup.sh**  \* \* \* \* \* command to be executed  - - - - -  | | | | |  | | | | |  | | | | +----- day of week (0 - 7) (Sunday=0 or 7)  | | | +---------- month (1 - 12)  | | +--------------- day of month (1 - 31)  | +-------------------- hour (0 - 23)  +------------------------- min (0 - 59)  Save your changes and exit the editor. Once you have modified your crontab, you should receive a message that a new crontab is being installed.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML15f78e5d.PNG |
|  | Wait for the time to pass for the job to execute and look for the backup files. Execute the following commands where you should see a second set of backups with a new timestamp:  [gpadmin@mdw ~]$ **ls -a $MASTER\_DATA\_DIRECTORY/db\_dumps/\***  [gpadmin@mdw ~]$ **gpssh -h sdw1 -h sdw2 \**  **ls -aR /data/primary/gpseg\*/db\_dumps**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1605648f.PNG  If the backup did not execute because time passed before you saved the crontab, you can execute the /home/gpadmin/cronbackup.sh script manually with the command,  /bin/bash /home/gpadmin/cronbackup.sh.  Do not move to the next step until the backup process has completed. |
|  | Over the next few steps, you will recover the database from the backup you created.  Connect to the gpadmin database as the gpadmin user, if not already connected:  [gpadmin@mdw ~]$ **psql**  Rename the faa database:  gpadmin=# **ALTER DATABASE faa rename to faa1;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML160a1e7b.PNG |
|  | Recreate the faa database. You will use this database to recover the data stored in the backup you created.  gpadmin=# **CREATE DATABASE faa;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML160b2fba.PNG  Connect to the faa database you created and list the tables in the database: gpadmin=# **\c faa** faa=# **\dt**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML160c9fbb.PNG |
|  | Exit your PSQL session. |
|  | Restore the faa database from the last backup taken. By not including the key identifier for the backupset, gpdbrestore will use the last available backup created to perform the restore operation.  Execute the gpdbrestore command to restore the faa database:  [gpadmin@mdw ~]$ **gpdbrestore -s faa**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML160e23c9.PNG  When prompted, type **y** and press Enter.  The procedure may take a few minutes to complete. |
|  | C:\Users\cantot\AppData\Local\Temp\SNAGHTML161ac2a6.PNG  **Note:** gpdbrestore –s *database\_name* option looks for the latest set of dump files for the given database name in the segment data directories db\_dumps directory on the Greenplum Database array of hosts. |
|  | Verify the database has been restored by listing tables from the faa database.  [gpadmin@mdw ~]$ **psql faa**  faa=# **ALTER DATABASE faa SET search\_path TO faadata, public, pg\_catalog;**  faa=# **\c faa**  faa=# **\dt**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML161c4f6b.PNG  C:\Users\cantot\AppData\Local\Temp\SNAGHTML161ca2e7.PNG |
|  | Exit the database before proceeding. |

Lab 15: Task 2 – Incremental Backup and Restore

| **Step** | **Action** |
| --- | --- |
|  | Execute the command below to create two tables in the dbbackupdatabase. The first will be a regular table called dimairline, while the second is an append-only table called dimairline\_image.  [gpadmin@mdw ~]$ **psql -f /rawdata/FAAData/CreateDbBackupTables.sql \ dbbackup**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML161fdb16.PNG  The error is displayed only if the table did not previously exist. The table will be created thereafter. |
|  | Start a psql session by connecting to thedbbackupdatabase.  [gpadmin@mdw ~]$ **psql dbbackup**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML162c4a5c.PNG |
|  | List the available tables on this database by using the command \dt as shown:  dbbackup=# **\dt dimairline\***  C:\Users\cantot\AppData\Local\Temp\SNAGHTML162cbf6c.PNG |
|  | Verify how many records the tables have by running the commands below.  dbbackup=# **select count(\*) from dimairline;**  dbbackup=# **select count(\*) from dimairline\_image;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML162d3b10.PNG  **Note:**  Both tables contain 1,540 records. |
|  | Exit the database before proceeding. |
|  | Run a full backup using the following command:  [gpadmin@mdw ~]$ **gpcrondump -x dbbackup**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML16309715.PNG  Respond with **y** when prompted to continue. |
|  | Connect to the dbbackup database as gpadmin and populate the append-only dimairline\_image table, as shown:  [gpadmin@mdw ~]$ **psql dbbackup**  dbbackup=# **COPY backupdata.DimAirline\_image FROM**  **'/rawdata/FAAData/DimAIRLINES.csv'**  **WITH DELIMITER ',' CSV HEADER QUOTE '"';**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1631b58d.PNG |
|  | Verify the number of records both the dimairline and dimairline\_image tables have by executing the following commands:  dbbackup=# **select count(\*) from dimairline;**  dbbackup=# **select count(\*) from dimairline\_image;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML16326b44.PNG  **Note:** The table dimairline\_image now contains 3,080 records. |
|  | Exit the database and start an incremental backup for the dbbackup database.  dbbackup=# **\q**  [gpadmin@mdw ~]$ **gpcrondump -x dbbackup --incremental**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1632eb4c.PNG  Respond with **y** when prompted to continue. |
|  | C:\Users\cantot\AppData\Local\Temp\SNAGHTML16358d07.PNG **Note:** The incremental backup provides a Dump key as shown that should be used to restore the incremental backup.  Record the dump key here:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Connect to the dbbackup database and truncate the dimairline and dimairline\_image tables.  [gpadmin@mdw ~]$ **psql dbbackup**  dbbackup=# **truncate table dimairline;**  dbbackup=# **truncate table dimairline\_image;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1636b281.PNG |
|  | Verify the number of records in the tables:  dbbackup=# **select count(\*) from dimairline;**  dbbackup=# **select count(\*) from dimairline\_image;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1637767b.PNG |
|  | Exit the database and execute the gpdbrestore command to recover the tables.  dbbackup=# **\q**  [gpadmin@mdw ~]$ **gpdbrestore -t *20150330150138***  Replace the dump key shown here with the dump key you recorded earlier.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1638e5ff.PNG  Respond with **y** when prompted to continue. |
|  | C:\Users\cantot\AppData\Local\Temp\SNAGHTML1639bf2e.PNG |
|  | Access the dbbackup database and verify the number of records the tables contain by executing the following commands:  [gpadmin@mdw ~]$ **psql dbbackup**  dbbackup=# **select count(\*) from dimairline;**  dbbackup=# **select count(\*) from dimairline\_image;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML163a8d45.PNG |
|  | Exit the database before proceeding. |
|  | **Summary**  Backups are typically automated with gpcrondump, which is a wrapper for gp\_dump and pg\_dumpall.  The gpcrondump utility dumps the contents of a Greenplum Database into SQL utility files, which can then be used to restore the database schema and user data at a later time using gpdbrestore.  Keep in mind that a database in the Greenplum Database is actually comprised of several PostgreSQL instances (the master and all active segments), each of which must be dumped individually. The gpcrondump utility takes care of dumping all of the individual instances across the system.  Note that the 14 digit timestamp is the number that uniquely identifies the backup job, and is part of the filename for each dump file created by a gp\_dump operation. This timestamp must be passed to the gpdbrestore utility when restoring a Greenplum Database.  Incremental backups let you backup append-only tables if a change has been made to the table or its contents. The --incremental option to the gpcrondump command lets you take advantage of the space-saving features that come with performing incremental backups on your tables. Restoring from an incremental backup requires that you have all backups from the last full backup. |

Lab 15: Task 3 – Recover from a Failed Master (OPTIONAL)

| **Step** | **Action** |
| --- | --- |
|  | **Task Overview**  Your standby server has been installed and configured during the installation of the Greenplum software.  You will perform the following steps:   * Failover to the standby server * Verify the Greenplum state operating with standby server * Failback to the master server |
|  | Before proceeding, verify that the standby server is properly configured.  If not already connected, log in as root and switch user to gpadmin on your master server.  Verify the file /etc/hostsonthe master and standby servers contain the same content.  [gpadmin@mdw ~]$ **cat /etc/hosts**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a3fe76b.PNG  [gpadmin@mdw ~]$ **gpssh -h smdw -e 'cat /etc/hosts'**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a405441.PNG |
|  | Verify the contents of the.bash\_profilefile on the standby mastser server are the same as the .bash\_profile file on the master server for the gpadmin user.  [gpadmin@mdw ~]$ **gpssh -h mdw -h smdw -e 'cat ~/.bash\_profile'**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a496706.PNG  As Command Center has not been configured to run on the standby server, you do not have to make changes to the /home/gpadmin/.bash\_profile file to include it. All other changes reflecting the Greenplum Database must be the same. |
|  | Verify that you can ssh to both segment servers from the standby server.  Open a new terminal connection to the standby server, smdw.  Connect to the standby server first. Login as root and switch to the gpadmin user account.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a50b43b.PNG |
|  | From the terminal session where you have connected to the standby server, connect to the first segment server, sdw1.  [gpadmin@smdw ~]$ **ssh sdw1**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a51adb6.PNG  Exit from the first segment server and connect to the second segment server, sdw2.  [gpadmin@sdw1 ~]$ **exit**  [gpadmin@smdw ~]$ **ssh sdw2**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a525431.PNG  Exit from the second segment server, sdw2.  [gpadmin@sdw2 ~]$ **exit**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a52bb4d.PNG |
|  | Before initiating a failover, verify the state of the master to standby server to ensure that the database is synchronized. To verify the state, execute the following command on the master server, mdw:  [gpadmin@mdw ~]$ **gpstate -f**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a56fb58.PNG |
|  | To safeguard against incidents that may occur in your lab environment, create a backup of all databases in the environment. You will create a backup of all databases except template0, template1, and postgres.  First, obtain the list of databases in the environment. The highlighted databases will be backed up.  [gpadmin@mdw ~]$ **psql -l**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1ab95fc7.PNG |
|  | Create a backup of all databases highlighted in the previous step using the gpcrondump command. You can specify multiple databases using the -x option with a comma separated list of databases. You will also copy the pogstresql.conf and pg\_hba.conf file as part of the backups with the -g option. The -a option will execute the command in non-interactive mode. Backup files will be saved to the /home/gpadmin/db\_backup directory on the master and segment servers with the -u option. You will need the configuration files after you complete the failover process.  [gpadmin@mdw ~]$ **gpcrondump -x dbbackup,dbstudent,faa,faa1,\ gpadmin,gpperfmon,names -u ~/db\_backup -g -a**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a625a69.PNG |
|  | From the master server, switch to the root user and issue the reboot command to reboot the master server. The database will not start automatically as there are no startup scripts in place for the database start up.  [gpadmin@mdw ~]$ **su -**  [root@mdw ~]# **reboot**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a6935ba.PNG  The purpose of this step is to simulate unavailability of the master server. You can then force the standby server to become the new primary master server.  Do not proceed until this step has been completed. |
|  | Confirm that your master server is down, by pinging it as shown from the standby masterserver.  [gpadmin@smdw ~]$ **ping mdw**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a6a9db0.PNG |
|  | From your standbyserver, smdw, promote the standby master to be the primary master. You will need to specify the port to use for the database activation. You will continue to use port 5432.  [gpadmin@smdw ~]$ **export PGPORT=5432**  [gpadmin@smdw ~]$ **gpactivatestandby -d /data/master/gpseg-1**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a781b96.PNG  Respond with **y** when asked to continue.  It may take a few minutes for the process to complete.  Note that the postgresql.conf and pg\_hba.conf files are not synchronized as part of the master replication process. Therefore, custom settings preserved on the master are not available here. This therefore required that you set the PGPORT environment variable before promoting the standby server to master. |
|  | Verify the state of your database by running the gpstate command with the -s option to obtain detailed information. Search the output for strings that contain the word, master.  [gpadmin@smdw ~]$ **gpstate –s | grep –i master**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a81f3bb.PNG  Note that there is no standby running at this point. |
|  | Open a PSQL session to the **faa1** database to verify the database has been recovered.  [gpadmin@smdw ~]$ **psql faa1**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a8316d5.PNG |
|  | Display the user tables for the faa1 database.  faa1=# **\dt**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a83c433.PNG |
|  | Exit the database. |
|  | After activating the standby server as the master server, you should update the database query statistics on all databases.  For each database, execute the ANALYZE command to update statistics. Use the following script to perform this step.  [gpadmin@smdw ~]$ **for db in \ `psql -tc "select datname from pg\_database where datistemplate='f';"`; do**  **echo -n "$db: "**  **psql $db -c 'ANALYZE';**  **done**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1b3df535.PNG |
|  | After activating a standby master in a recovery scenario and making it your current primary master, you can continue running that instance as your primary master. This assumes that the capabilities and dependability of that host machine are equivalent to the original master host.  Before restoring the master and standby instances on original hosts, ensure that the conditions that caused the original failure have been fully fixed.  Verify the original master server, mdw, is back online. Ping mdw from smdw.  [gpadmin@smdw ~]$ **ping mdw**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a88069f.PNG |
|  | Reconnect your original terminal session to the original master server, mdw. Login as root and switch to the gpadmin user account.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a89b4e8.PNG |
|  | On your original master server, mdw, rename the directory /data/master/gpseg-1 to /data/master/gpseg-1\_orig. The utility gpinitstandby will recreate the directory and requires that it does not exist.  [gpadmin@mdw ~]$ **mv $MASTER\_DATA\_DIRECTORY \ ${MASTER\_DATA\_DIRECTORY}\_orig**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1a9acde4.PNG |
|  | From the standby server, smdw,execute the gpinitstandby command to promote the original master server, mdw, to be the new standby server.  [gpadmin@smdw ~]$ **gpinitstandby -s mdw**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1b409339.PNG  You will be prompted to continue. Type **y** and press Enter to proceed.  Once completed, you should see the following output:  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1b429d47.PNG |
|  | Use the gpstate command to check the status of the standby master. The output of the gpstate command shows that the original standby server, smdw**,** is now the master server. It also shows that the original master server, mdw, is now the standby server.  [gpadmin@smdw ~]$ **gpstate -s | grep -i master**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1b43695b.PNG  You can also obtain the state of the standby and mirrors with the command, **gpstate -f**. |
|  | Now that the failover has succeeded, reverse the roles of the master and standby servers so that mdw and smdw are back to their original roles. To perform this task, complete the following steps:  On the current master server, smdw, stop the Greenplum database master instance only using the -am option.  [gpadmin@smdw ~]$ **gpstop -am**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1b95ec04.PNG |
|  | At this point, the database should no longer be running. From the current standby server, mdw, promote mdw to be the active master server. Use the gpactivatestandby utility to perform this task**.**  [gpadmin@mdw ~]$ **export PGPORT=5432**  [gpadmin@mdw ~]$ **gpactivatestandby -d $MASTER\_DATA\_DIRECTORY -f**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1b5156a6.PNG  You will be prompted to continue. Type **y** and press Enter to proceed.  Once completed, you should see the following output:  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1b9c342a.PNG |
|  | On mdw, execute the gpstate utility to determine the state of the active master server. This screen shows that there is no standby server configured and that mdw is back to its original role as the active master server.  [gpadmin@mdw ~]$ **gpstate -s | grep -i master**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1b560da6.PNG |
|  | On smdw, start the process for changing this server back to its original role as the standby server.  Rename the directory /data/master/gpseg-1 to /data/master/gpseg-1\_orig.  [gpadmin@smdw ~]$ **mv $MASTER\_DATA\_DIRECTORY \ ${MASTER\_DATA\_DIRECTORY}\_orig**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1b57d779.PNG  This step is necessary as the procedure to initialize a standby server into the Greenplum cluster will create the master data directory. |
|  | From mdw, execute the gpinitstandby utility to promote the smdw server to the standby role.  [gpadmin@mdw ~]$ **gpinitstandby -s smdw**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1b59e020.PNG  You will be prompted to continue. Type **y** and press Enter to proceed. |
|  | Run the gpstate utility to re-verify the state of the master and standby servers. The screen below shows that the original roles for mdw (master) and smdw (standby) have been restored.  [gpadmin@mdw ~]$ **gpstate -s | grep -i master**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1b5c2f97.PNG |
|  | Recover the postgresql.conf and pg\_hba.conf file from your backups and push them to the $MASTER\_DATA\_DIRECTORY locations. This will overwrite the existing copies but will give you back your customized versions.  Search the backup directory you created in this task for files with the name gp\_master\_config\_files\_\*.tar.  [gpadmin@mdw ~]$ **find ~/db\_backup -name 'gp\_master\_config\_files\_\*.tar'**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1b75eea2.PNG |
|  | Extract the contents of the tarred file from the previous step.  [gpadmin@mdw ~]$ **tar xvf /home/gpadmin/db\_backup/db\_dumps/20150331/gp\_master\_config\_files\_20150331103817.tar**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1b788593.PNG |
|  | Copy the pg\_hba.conf file and the postgresql.conf file to $MASTER\_DATA\_DIRECTORY.  [gpadmin@mdw ~]$ **cp data/master/gpseg-1/pg\_hba.conf $MASTER\_DATA\_DIRECTORY**  [gpadmin@mdw ~]$ **cp data/master/gpseg-1/postgresql.conf $MASTER\_DATA\_DIRECTORY**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1b7acb79.PNG |
|  | Re-read the configuration files with the gpstop -u command.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1b7b5abc.PNG |
|  | As a test, verify your non-superuser account, in this case student, can list the tables in the faa database.  [gpadmin@mdw ~]$ **psql faa -c '\dt' -U student**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1b7de781.PNG |
|  | **Summary**  If the master server fails, the standby server can be used to bring the database back online and accessible to users. If a virtual IP address has been defined for the master and standby server, the virtual IP address can be used by the standby server so that users do not have to use a different IP address or hostname to access the database.  Greenplum Database uses log replication to synchronize data between the master server and the backup server. Committed transactions are synchronized from the master server to the standby server. Should the master become unavailable, the standby can be promoted to act as the master until the master becomes available again.  The replication process is maintained by a WAL process running on the master server and the standby server. You can verify the state of synchronization by using the gpstate -f command or selecting against the pg\_stat\_replication view. This view contains the process id (procpid field), the state (state field), and the synchronization state (sync\_state field) along with other information on the WAL process.  Note that while transactions are synchronized, the postgresql.conf and pg\_hba.conf files are not. Maintain a backup copy of these files and be prepared to incorporate the changes to those files on the standby server should you need to perform a failover.  Additionally, when promoting the original master server to its original role, retrieve the backup copies of those files and push them to the recovered data directory. |

*End of Lab Exercise*

This page intentionally left blank

# Lab Data Modeling

|  |  |
| --- | --- |
| **Purpose:** | In this lab, you create the database objects that will be used for follow-on labs for demonstrating data modeling and design decisions. |
|  | |
| **Tasks:** | Students perform the following task: Create the datamart database and database objects. |
|  | |
| **References:** | Module 7: Data Modeling and Design   * Lesson: Data Modeling |

Lab 16: Task – Create the datamart Database and Objects

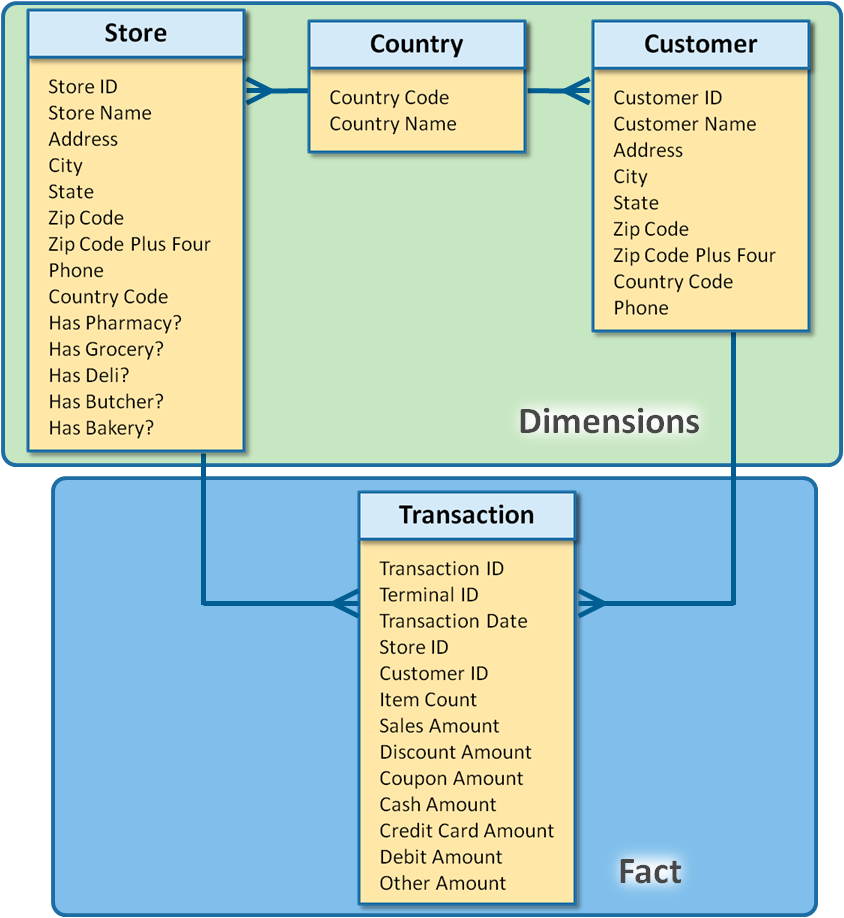
| **Step** | **Action** |
| --- | --- |
|  | Create the datamart database using either of the methods discussed.  From the Linux command prompt:  [gpadmin@mdw ~]$ **createdb datamart**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f4dcfef.PNG |
|  | Access the datamart database as gpadmin:  [gpadmin@mdw ~]$ **psql datamart**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f4e39aa.PNG |
|  | Create the facts and dimensions schema for the database:  datamart=# **CREATE SCHEMA dimensions;** datamart=# **CREATE SCHEMA facts;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f4eb011.PNG |
|  | Set the default search path for the datamart database to include the facts and dimensions schemas. You may wish to include public and pg\_catalog as well.  datamart=# **ALTER DATABASE datamart SET search\_path=dimensions,facts,public,pg\_catalog;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f505b3d.PNG |

*End of Lab Exercise*

This page intentionally left blank

# Lab Physical Design Decisions

|  |  |
| --- | --- |
| **Purpose:** | In this lab, you create table objects based on business requirements provided in the lab. The choice of data types, constraints, distribution keys, and partitioning is up to you.  The logical data model to support the business model is provided below. |
|  | |
| **Tasks:** | Students perform the following tasks:   * + Create the Store dimension   + Create the Country dimension   + Create the Customer dimension   + Create the Transaction fact table   + Load dimension and fact data |
|  | |
| **References:** | Module 7 – Data Modeling and Design   * Lesson: Physical Design Decisions |



Lab 17: Task 1 – Create the Store Dimension

| **Step** | **Action** |
| --- | --- |
|  | The business requirements for the Store dimension are as follows:   1. Each store is given a numeric identifier. This identifier is created in the source system (OLTP) and passed to the data warehouse. The Store id is unique for each store and may not be null. 2. All address attributes are required. 3. The phone number is stored as (XXX)XXX-XXXX formatted character string. 4. The store name must allow for upper and lower case data and may contain single or double quotes in the name. The store name may not be null. 5. The “Has” columns are passed as either 0 (false) or 1 (true) to the data warehouse. The default value for these columns in the OLTP system is 0. 6. The country code is three characters (e.g., USA, CAN). Currently the business only operates in the United States and Canada. 7. There are currently around thirty five stores in the US and Canada.   Based on these requirements, implement the Store dimension. |
|  | Determine the table name for the Store entity?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Determine the column name, data type, and any constraints for each attribute of the Store entity.   |  |  |  |  | | --- | --- | --- | --- | | Attribute Name | Column Name | Data Type | Constraint | | Store ID |  |  |  | | Store Name |  |  |  | | Address |  |  |  | | City |  |  |  | | State |  |  |  | | Zip Code |  |  |  | | Zip Code Plus Four |  |  |  | | Phone |  |  |  | | Country Code |  |  |  | | Has Pharmacy |  |  |  | | Has Grocery |  |  |  | | Has Deli |  |  |  | | Has Butcher |  |  |  | | Has Bakery |  |  |  | |
|  | What will be the DISTRIBUTION KEY? Is it UNIQUE? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Create an SQL file with the o statement.  Be sure to include any column and table constraints.  There is an example (non-optimized) file called create\_store.sql in the /home/gp/sql/load\_files /adv\_sql\_files that you can use as a guide.  [gpadmin@mdw ~]$ **cd /home/gp/sql/load\_files/adv\_sql\_files**  [gpadmin@mdw adv\_sql\_files]$ **vi create\_store.sql**  You will be accessing the directory /home/gp/sql/load\_files directory throughout this lab. To save some time, create an environment variable that points to that directory: **export LF=/home/gp/sql/load\_files** |
|  | Execute the script as gpadmin by starting a **psql** session and passing in the file name with the **-f** option. Include the datamart database as part of the psql command:  [gpadmin@mdw adv\_sql\_files]$ **psql datamart -f create\_store.sql**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f6fb436.PNG  You should see the words CREATE TABLE immediately under the Linux prompt if there were no errors. |
|  | Verify the table has been created:  [gpadmin@mdw adv\_sql\_files]$ **psql datamart -c "\dt dimensions.store"**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f7020fc.PNG  One row with the table name should be displayed. |

Lab 17: Task 2 – Create the Country Dimension

| **Step** | **Action** |
| --- | --- |
|  | The business requirements for the Country dimension are as follows:   1. The country code is three characters, such as USA or CAN. Currently, the business only operates in the United States and Canada. This column cannot be NULL. 2. The country name is a text column.   Based on these requirements, implement the Country dimension. |
|  | Determine the table name for the Country entity from the LDM. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Determine the column name, data type and any constraints for each attribute of the Store entity.   |  |  |  |  | | --- | --- | --- | --- | | Attribute Name | Column Name | Data Type | Constraint | | Country Code |  |  |  | | Country Name |  |  |  | |
|  | What will be the DISTRIBUTION KEY? Is it UNIQUE? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Create an SQL file with the CREATE TABLE statement.  Be sure to include any column and table constraints.  There is an example (non-optimized) file called create\_country.sql in the /home/gp/sql/load\_files /adv\_sql\_files that you can use as a guide.  [gpadmin@mdw adv\_sql\_files]$ **vi create\_country.sql** |
|  | Execute the script as gpadmin by starting a psql session and passing in the file name with the -f option. Include the datamart database as part of the psql command:  [gpadmin@mdw adv\_sql\_files]$ **psql datamart -f create\_country.sql**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f70cc29.PNG  You should see the words CREATE TABLE immediately under the UNIX prompt if there were no errors. |
|  | Verify the table has been created:  [gpadmin@mdw adv\_sql\_files]$ **psql datamart \ -c "\dt dimensions.country"** C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f711fc4.PNG  One row with the table name should be displayed. |

Lab 17: Task 3 – Create the Customer Dimension

| **Step** | **Action** |
| --- | --- |
|  | The business requirements for the Customer dimension are as follows:   1. Each customer is tracked by their phone number as part of their membership rewards program. 2. There is a default customer with a (999)999-9999 customer id that is used when a transaction occurs for a customer that is not a member of the rewards program. 3. The customer name may not be NULL. 4. All address attributes are required. 5. The phone number is stored as (XXX)XXX-XXXX formatted character string. 6. The country code is three characters (e.g., USA, CAN). Currently the business only operates in the United States and Canada. 7. There are approximately 10,000 customers signed up for the rewards program. 8. The customer id will have to be generated during the ETL process as it is not a part of the OLTP system.   Based on these requirements, implement the Customer dimension. |
|  | Determine the table name for the Customer entity from the LDM. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Determine the column name, data type and any constraints for each attribute of the Store entity.   |  |  |  |  | | --- | --- | --- | --- | | Attribute Name | Column Name | Data Type | Constraint | | Customer ID |  |  |  | | Customer Name |  |  |  | | Address |  |  |  | | City |  |  |  | | State |  |  |  | | Zip Code |  |  |  | | Zip Code Plus Four |  |  |  | | Phone |  |  |  | | Country Code |  |  |  | |
|  | What will be the DISTRIBUTION KEY? Is it UNIQUE? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Create an SQL file with the CREATE TABLE statement.  Be sure to include any column and table constraints.  There is an example (non-optimized) file called create\_customer.sql in the /home/gp/sql/load\_files /adv\_sql\_files that you can use as a guide.  [gpadmin@mdw adv\_sql\_files]$ **vi create\_customer.sql** |
|  | Execute the script as gpadmin by starting a psql session and passing in the file name with the -f option. Include the datamart database as part of the psql command:  [gpadmin@mdw adv\_sql\_files]$ **psql datamart -f create\_customer.sql**    You should see the words CREATE TABLE immediately under the UNIX prompt if there were no errors. |
|  | Verify that the table has been created:  [gpadmin@mdw adv\_sql\_files]$ **psql datamart \ -c "\dt dimensions.customer"**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f731b23.PNG |

Lab 17: Task 4 – Create the Transaction Fact Table

| **Step** | **Action** |
| --- | --- |
|  | The business requirements for the Transaction fact table are as follows:   1. The transaction represents all transactions that occur by a customer, at a checkout stand, at a given store by day. (In data warehousing this is called the “shopping basket” type of transaction.) It is, in effect, a summary of the all items and includes tax, payment methods, etc. 2. Each transaction is given a numeric identifier. This identifier is created in the source system (OLTP) and passed to the data warehouse. The transaction id is unique for each customer transaction and may not be null. 3. If the customer is not a rewards member, the default phone number is (999)999-9999. Otherwise rewards customers are tracked by their phone numbers. 4. The terminal id may not be null. All terminals are numbered from 1-N in each store. There are no more than 24 terminals in any given store at implementation time. 5. Item count may not be zero or negative. 6. Sales Amount may not be negative. 7. Taxes are represented in the “other amount” attribute. 8. Cash Amount + Debit Amount + Credit Card Amount may not exceed Sales Amount plus Other Amount. 9. There is an average of 100,000 transactions per day across the entire business. 10. Ninety percent of all transactions involve a rewards member customer. 11. The transaction date is a timestamp containing the date with the time. The business only wishes to store the date portion of the transaction date. 12. The historical data that will be loaded only goes back to the beginning of 2008. |
|  | Determine the table name for Transaction entity. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Determine the column name, data type, and any constraints for each attribute of the Transaction entity.   |  |  |  |  | | --- | --- | --- | --- | | Attribute Name | Column Name | Data Type | Constraint | | Transaction ID |  |  |  | | terminal ID |  |  |  | | Transaction Date |  |  |  | | Store ID |  |  |  | | Customer ID |  |  |  | | Item Count |  |  |  | | Sales Amount |  |  |  | | Discount Amount |  |  |  | | Coupon Amount |  |  |  | | Cash Amount |  |  |  | | Credit Card Amount |  |  |  | | Debit Amount |  |  |  | | Other Amount |  |  |  | |
|  | What will be the DISTRIBUTION KEY? Is it UNIQUE?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Create an SQL file with the CREATE TABLE statement.  Be sure to include any column and table constraints.  There is an example (non-optimized) file called create\_ transaction.sql in the /home/gp/sql/load\_files /adv\_sql\_files that you can use as a guide.  [gpadmin@mdw adv\_sql\_files]$ **vi create\_transaction.sql** |
|  | Execute the script as gpadmin by starting a psql session and passing in the file name with the -f option. Include the datamart database as part of the psql command:  [gpadmin@mdw adv\_sql\_files]$ **psql datamart -f create\_transaction.sql**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f739715.PNG |
|  | Verify the table has been created:  [gpadmin@mdw adv\_sql\_files]$ **psql datamart -c "\d+ facts.transaction"**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f73df5b.PNG |

Lab 17: Task 5 – Load Dimension and Fact Data

| **Step** | **Action** |
| --- | --- |
|  | Connect to the datamart database as gpadmin.  [gpadmin@mdw adv\_sql\_files]$ **psql datamart** |
|  | Load the dimensions.country table using the COPY command. Copy the data from /home/gp/sql/load\_files/CountryData.csv into the dimensions.country table.  datamart=# **COPY dimensions.country**  **FROM '/home/gp/sql/load\_files/CountryData.csv'**  **WITH CSV HEADER**  **LOG ERRORS INTO public.country\_err**  **SEGMENT REJECT LIMIT 10 ROWS;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f76146f.PNG |
|  | Check the error table, public.country\_err for any errors:  datamart=# **SELECT \* FROM public.country\_err;**  Were any rows discarded? How many? Why?  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f768173.PNG  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Load the dimensions.store table using the COPY command. Copy the data from /home/gp/sql/load\_files/StoreData.csv into the dimensions.store table.  datamart=# **COPY dimensions.store**  **FROM '/home/gp/sql/load\_files/StoreData.csv'**  **WITH CSV HEADER**  **LOG ERRORS INTO public.store\_err**  **SEGMENT REJECT LIMIT 10 ROWS;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f76f06b.PNG |
|  | Check the error table, **public.store\_err** for any errors:  datamart=# **SELECT \* FROM public.store\_err;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f77f7ea.PNG  Were any rows discarded? How many? Why?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Connect to the **datamart** database as **gpadmin**.  [gpadmin@mdw adv\_sql\_files]$ **psql datamart** |
|  | Load the dimensions.customer table using an external table. Create the external table using the following syntax:  datamart=# **CREATE EXTERNAL TABLE public.customer\_external (**  **custName VARCHAR(50),**  **address VARCHAR(50),**  **city VARCHAR(40),**  **state CHAR(2),**  **zipcode CHAR(8),**  **zipPlusFour CHAR(4),**  **countrycd CHAR(3),**  **phone CHAR(13)**  **)**  **LOCATION ('file://sdw1/home/gp/sql/load\_files/CustomerData.csv')**  **FORMAT 'CSV' (HEADER DELIMITER ',')**  **LOG ERRORS INTO public.customer\_err**  **SEGMENT REJECT LIMIT 10 ROWS;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f793f07.PNG |
|  | Validate that the table was created correctly:  datamart=# **SELECT \* FROM public.customer\_external;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f79dc30.PNG |
|  | The customers for this business are tracked by their phone numbers. A customer ID must be created for each of the customers. You will use a sequence number to create a unique customer ID for each customer. Create a sequence to support the data load of the customer dimension:  datamart=# **CREATE SEQUENCE public.CustomerSequence**  **INCREMENT BY 1**  **START WITH 1**  **NO CYCLE;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f7a435b.PNG |
|  | Insert the rows from the external table into the target table. Note that you are performing a transform of the ELT process, so you will need to specify all of the columns from the source table. Note that the sequence name is all lower case.  datamart=# **INSERT INTO dimensions.customer**  **(CustomerID,**  **custName,**  **address,**  **city,**  **state,**  **zipcode,**  **zipPlusFour,**  **countrycd, phone)**  **SELECT nextval('customersequence'),**  **custName,**  **address,**  **city,**  **state,**  **zipcode,**  **zipPlusFour,**  **countrycd,**  **phone**  **FROM public.customer\_external;** C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f7adeb0.PNG |
|  | Were any rows discarded? How many? Why?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Exit the PSQL session. |
|  | Load the Transaction data using gpfdist and an external table. You will perform a single transformation during the INSERT… SELECT and that is a join to the Customer dimension using the phone column to get the customerid.  Start a gpfdist session and verify that it is executing:  [gpadmin@mdw adv\_sql\_files]$ **gpfdist -d /home/gp/sql/load\_files \ -p 8081 -l /tmp/gpfdist.log &**  Verify that gpfdist is running.  [gpadmin@mdw adv\_sql\_files]$ **ps -ef | grep gpfdist**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f7d8912.PNG  **Note:** If you receive an internal error that gpfdist cannot create a socket, verify that another gpfdist process is not running on the port number you specified. |
|  | Connect to the datamart database as gpadmin.  [gpadmin@mdw adv\_sql\_files]$ **psql datamart** |
|  | Create a gpfdist based external table to load the Transaction fact table. Connect to the datamart database and use the following DDL to create the external table:  datamart=# **CREATE EXTERNAL TABLE public.transaction\_external**  **(transId BIGINT,**  **terminalId INTEGER,**  **transDate DATE,**  **storeId SMALLINT,**  **phone CHAR(14),**  **itemCnt INTEGER,**  **salesAmt DECIMAL(9,2),**  **taxAmt DECIMAL(9,2),**  **discountAmt DECIMAL(9,2),**  **couponAmt DECIMAL(9,2),**  **cashAmt DECIMAL(9,2),**  **checkAmt DECIMAL(9,2),**  **ccAmt DECIMAL(9,2),**  **debitAmt DECIMAL(9,2),**  **otherAmt DECIMAL(9,2)**  **)**  **LOCATION**  **('gpfdist://mdw:8081/TransactionData001.csv',**  **'gpfdist://mdw:8081/TransactionData002.csv')**  **FORMAT 'csv' (HEADER DELIMITER '|')**  **LOG ERRORS INTO public.transaction\_err**  **SEGMENT REJECT LIMIT 10 ROWS;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f7ee9d6.PNG |
|  | Validate that the table was created correctly:  datamart-# **SELECT \* FROM public.transaction\_external;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f7f4619.PNG |
|  | Insert the rows from the external table into the target Transaction fact table in psql. Join the external transaction table to the customer dimension using the phone number column to get the customer id. If there were hundreds of thousands or millions of rows in the source external files, you should consider loading a specific child table or using gpload.  datamart=# **INSERT INTO facts.transaction**  **(transid,**  **terminalid,**  **transdate,**  **storeid,**  **customerid,**  **itemcnt,**  **salesamt,**  **taxamt,**  **discountamt,**  **couponamt,**  **cashamt,**  **checkamt,**  **ccamt,**  **debitamt,**  **otheramt)**  **SELECT t.transid, t.terminalid, t.transdate, t.storeid,**  **c.customerid, t.itemcnt, t.salesamt, t.taxamt,**  **t.discountamt, t.couponamt, t.cashamt, t.checkamt, t.ccamt,**  **t.debitamt, t.otheramt**  **FROM public.transaction\_external t**  **INNER JOIN dimensions.customer c**  **ON TRIM(c.phone) = TRIM(t.phone);**C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f8051fb.PNG |
|  | Change back to the home directory.  [gpadmin@mdw adv\_sql\_files]$ **cd** |
|  | Were any rows discarded? How many? Why?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Why did you need the TRIM function? What are the implications for loading large rows where we need to apply a function to the join conditions?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

*End of Lab Exercise*

# Lab Database Tuning

|  |  |
| --- | --- |
| **Purpose:** | In this lab, you use a combination of tools, commands, and schemas to analyze the overall performance of your environment. You will examine how the system behaves when retrieving data or updating tables. |
|  | |
| **Tasks:** | Students perform the following task: Tune the database and queries. |
|  | |
| **References:** | Module 8 – Performance Analysis and Tuning   * Lesson: Join Tables – Types and Methods * Lesson: Database Tuning * Lesson: Query Profiling |

Lab 18: Task – Tune the Database and Queries

| **Step** | **Action** |
| --- | --- |
|  | While logged in as gpadminon your master server, verify your parameters are as shown below.  [gpadmin@mdw ~]$ **gpconfig -s *parameter***  Verify the following parameters:  max\_fsm\_relations = 1000  max\_fsm\_pages = 200000  work\_mem = 32MB  maintenance\_work\_mem = 64MB  For example, to search for max\_fsm\_relations, type the following:  [gpadmin@mdw ~]$ **gpconfig -s max\_fsm\_relations**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1f84e026.PNG |
|  | Load data in a table and check performance. Connect to the faa database as gpadmin.  [gpadmin@mdw ~]$ **psql faa** |
|  | The PSQL timing parameter shows how long it takes to complete a command. The command will be used to compare the time it takes to execute SELECT statements on specific tables.  Turn timing on:  faa=# **\timing**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1ff55190.PNG |
|  | Create a new table and load as shown. This step will create a new table and load it with more than 20 million records thus it might take a couple of minutes.  faa=# **create table factontimeperformance2 as select \* from factontimeperformance;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1fdf3e10.PNG |
|  | Verify the data loaded:  faa=# **SELECT count(\*) from factontimeperformance2;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1fe2a21f.PNG |
|  | Analyze and vacuum the table:  faa=# **vacuum analyze factontimeperformance2;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1ff0155b.PNG |
|  | Re-check performance:  faa=# **SELECT count(\*) from factontimeperformance2;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1ff10516.PNG  Note that the time it took to perform this query should be less than the first time the query was executed. The VACUUM ANALYZE step you performed provided more up to date statistical information to the query planner so that it could determine the most optimal plan for the query. |
|  | Disable PSQL timing. Use the \timing command to toggle the setting to off.  faa=# **\timing**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1ff5f741.PNG |
|  | Greenplum provides an administrative schema called gp\_toolkit that you can use to query the system catalogs, log files, and operating environment for system status information. The gp\_toolkit schema contains a number of views that you can access using SQL commands. The gp\_toolkit schema is accessible to all database users, although some objects may require superuser permissions.  Look for tables that do not have statistics:  faa=# **SELECT \* from gp\_toolkit.gp\_stats\_missing;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1ff6bf7f.PNG |
|  | Look for table bloat from lack of vacuuming:  faa=# **SELECT \* from gp\_toolkit.gp\_bloat\_diag limit 5;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1ff71f4a.PNG |
|  | If there is performance issue based on how a table is distributed, the distribution can be changed with an ALTER TABLE command:  faa=# **ALTER TABLE factontimeperformance SET DISTRIBUTED BY (year);**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1ffd8fa9.PNG  **Note:** This command may take several minutes to process. |
|  | Examine other views such as the pg\_stat\_activity, pg\_locks and pg\_class views.  pg\_stat\_activity has information about current running queries.  Look for the start time of all current queries in the database:  faa=# **select query\_start, procpid from pg\_stat\_activity;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML1ffe57c8.PNG |
|  | pg\_locks has data about locks in the database. This syntax shows the locks that are on the tables being accessed.  faa=# **select datname, relname, pid, mode**  **from pg\_locks,pg\_database,pg\_class**  **where pg\_locks.database=pg\_database.oid and**  **pg\_locks.relation=pg\_class.oid;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML200bb2c3.PNG  In this case, the tables being accessed are the ones referenced in the query just executed. There are no other accesses on the database at this time.Look for locks in the database: |
|  | Look for owner and object type in the database:  faa=# **select relname, relowner from pg\_class limit 5;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML20100e5a.PNG |
|  | Exit your PSQL session. |
|  | Database logs are another place to look for errors or tuning issues. In UNIX as the gpadmin user  The following are the database log files that reside on the system:   * + pg\_xlog contains Greenplum Write Ahead Logs (WAL, Greenplum implementation of transaction logging) files (normally 16MB in size, each).   + pg\_clog contains the commit log files which contain transaction commit status of a transaction. One main purpose is to perform a database recovery in case of a crash by replaying these logs.   + pg\_log contains the database instance logs by date. This is where you will find instance errors like missing files or filled up disks. It also holds the connection information.   Identify the logs that exist and use gplogfilter to access the content. The gplogfilter syntax here is looking at one file instead of all log files (the default behavior) and looks for the last three error messages generated in the file you specify.  [gpadmin@mdw ~]$ **ls $MASTER\_DATA\_DIRECTORY/pg\_log**  gpdb-2015-03-31\_160631.csv gpdb-2015-04-01\_000000.csv startup.log [gpadmin@mdw ~]$ **gplogfilter $MASTER\_DATA\_DIRECTORY/pg\_log/gpdb-2015-04-01\_000000.csv -n 3 –t**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML201d7a07.PNG  **Note:** Replace the file name shown with a file that exists in your pg\_log environment. |
|  | Use UNIX operating system commands to see performance from the OS point of view:   * + top is used to get information on CPU and memory performance.   + df provides information on the capacity of the file systems.   + ps -ef | grep postgres lets you look for postgres processes currently running.   + vmstat is a utility that provides information on virtual memory usage on the system.   + netstat is a network status utility.   + gpstate is a Greenplum utility that provides information on the state of the cluster. |
|  | Getting an explain plan out of a query in Greenplum is easy. Add the word EXPLAIN to the front of your query to see how the query optimizer will execute your query. You can then change the query to see if you can affect the cost to make it lower.  Open another PSQL session and run the two queries below. Verify the differences in the EXPLAIN plan:  faa=# **EXPLAIN select flightnum, dayid**  **from factontimeperformance, dimairline, dimairport**  **where dimairline.airlinename = 'United Air Lines Inc.: UA' and**  **dimairport.airportdescription = 'Denver, CO: Denver International'**  **and factontimeperformance.airlineid = dimairline.airlineid**  **and dimairport.airportid = factontimeperformance.originairportid;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML202240bf.PNG |
|  | faa=# **EXPLAIN select flightnum, dayid**  **from factontimeperformance, dimairline, dimairport**  **where dimairport.airportid = factontimeperformance.originairportid**  **and dimairline.airlinename = 'United Air Lines Inc.: UA'**  **and dimairport.airportdescription = 'Denver, CO: Denver International'**  **and factontimeperformance.airlineid = dimairline.airlineid;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML20237c68.PNG  **Note:** Both of the queries result in the same execution plan. Based on the statistics Greenplum will optimize the query the same way regardless of the ordering of the WHERE clause. |
|  | Generate execution plans for the following two queries and compare the execution plans:  faa=# **EXPLAIN SELECT distinct carrierid, flightnum FROM factontimeperformance f, dimairline al, dimairport ap WHERE f.airlineid = al.airlineid   AND f.originairportid = ap.airportid  AND f.originairportid in (SELECT AIRPORTID  FROM dimairport  WHERE airportdescription = 'Denver, CO: Denver International')  AND f.destairportid in (SELECT AIRPORTID  FROM dimairport  WHERE airportdescription = 'Boston, MA: Logan International');**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML2024635c.PNG |
|  | faa=# **EXPLAIN SELECT distinct carrierid, flightnum FROM factontimeperformance f, dimairline al, dimairport ap WHERE f.airlineid = al.airlineid   AND f.originairportid in (SELECT AIRPORTID  FROM dimairport  WHERE airportdescription = 'Denver, CO: Denver International')  AND f.destairportid in (SELECT AIRPORTID  FROM dimairport  WHERE airportdescription = 'Boston, MA: Logan International');**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML20250278.PNG  **Note:** While both of the queries display the same results, they have very different execution plans. The second query omits one of the join relations and creates a partial Cartesian product which has much slower performance. This can be identified in the execution plan from the additional Materialize step that is required. |
|  | Exit your PSQL session. |
|  | **Summary**  The EXPLAIN command allows you to view the query plan for a query. EXPLAINANALYZE will actually run the query and show you the plan that was executed but does not return results.  Query plans are read from bottom to top and show a tree plan of nodes. A node represents a database operation, such as a table scan, a join, or a sort. Greenplum Database plans will also show motion nodes, which are operations that move tuples between the segment instances or from the segment instances to the master. Examining query plans helps uncover areas where performance can be improved.  In addition to query plans, using UNIX-based commands and tools, Greenplum clients can provide detailed information on how the system is behaving. The hardware and network has a very strong impact on performance, so those should always remain a part of your performance tuning goals. |

*End of Lab Exercise*

# Lab Explain the EXPLAIN Plan – Analyzing Queries

|  |  |
| --- | --- |
| **Purpose:** | In this lab, you will be given a set of queries to analyze. The data that you have previously inserted deliberately does not have statistics collected for it. Do not use the EXPLAIN or ANALYZE commands until you are instructed to do so. You will want to compare the before and after effects of statistics. |
|  | |
| **Tasks:** | Students perform the following task: Analyze queries. |
|  | |
| **References:** | Module 8 – Performance Analysis and Tuning   * Explain the Explain Plan – Analyzing Queries |

Lab 19: Task 1– Configure Databases in pgAdmin III

| **Step** | **Action** |
| --- | --- |
|  | pgAdmin III offers a graphical interface to interact with the Greenplum Database. The tool can be used to view database objects, execute queries, and perform some management tasks in the database. You will configure pgAdmin III to make it easier to execute and the labs in this exercise. Feel free to use it for other exercises.  You must first configure pgAdmin III by completing two main tasks:   * Updating the pg\_hba.conf file to provide access to connections from your Windows / Mac environment where you will be running pgAdmin III * Creating a connection to the database within pgAdmin III |
|  | From your terminal connection the master server, mdw, modify the **$MASTER\_DATA\_DIRECTORY/pg\_hba.conf** file and add the following entry to the bottom of the file:  **host all gpadmin 172.16.1.15/32 trust**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML203c04e9.PNG  Save and exit the file. |
|  | Re-read the configuration file with the **gpstop** command.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML203d0d13.PNG |
|  | Double-click the pgAdmin III icon on your desktop to start the application. |
|  | Click the icon to add a connection to the environment.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML2035968d.PNG |
|  | Provide the following information to register your server connection and click **OK** to complete the process:   * Name: training * Host: 172.16.1.11 * Port: 5432 * Password: gpadmin |
|  | A warning will be displayed emphasizing the issues with storing your password. In your production environment, you would not do this. However, in this lab environment, you will proceed as it simplifies and eases your access.  Click **OK** to proceed. |
|  | The session will now be saved. Double-click the session under the Servers category to initiate a connection to the database.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML2041eac4.PNG  You now have access to the database through pgAdmin III. |

Lab 19: Task 2 – Analyze Queries

| **Step** | **Action** |
| --- | --- |
|  | You can choose to use pgAdmin III to view the explain plans for the queries you will examine in this task. Instructions will be provided for both for the first exercise. Remaining exercises will use PSQL. However, feel free to use pgAdmin III to view the query plans or execute queries. |
|  | If you wish to view the query plans in PSQL, proceed to step 8.  In pgAdmin III, click on the **datamart** database. This will be the database used for query executions.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML205e6706.PNG |
|  | Click the Execute Arbitrary SQL queries icon to proceed.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML205b0a75.PNG |
|  | Click the **Open file** icon.  C:\Users\cantot\AppData\Local\Temp\SNAGHTML20601d88.PNG |
|  | From the Documents folder, select the file, gp 🞂sql 🞂 load\_files 🞂 adv\_sql\_files 🞂 lab5query1 and either double-click the filename or click **Open**. |
|  | The first line of the query is the explain command. Remove this line.    While you can leave this line in the file and use the Execute query button to view the text of the explain plan, we will instead use the pgAdmin III explain plan method to view a graphical breakdown of the plan. |
|  | Click the **Explain query** icon to view the explain plan. |
|  | The Output pane displays a graphical analysis of the query in the Explain tab.    You can also view a textual analysis by clicking the Data Output tab.    Proceed to step 10. |
|  | Complete the following steps to view the explain plan in PSQL.  Access the datamart database as gpadmin.  [gpadmin@mdw ~]$ **psql datamart** |
|  | Execute an explain plan for a simple query without analyzing the data. This query can be found in the file /home/gp/sql/load\_files/adv\_sql\_files/lab5query1.sql.  datamart=# **\i /home/gp/sql/load\_files/adv\_sql\_files/lab5query1.sql**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML20728c51.PNG  The screenshot shows the text of the explain plan being executed. You are executing the query directly from the file provided. |
|  | Which tables are being scanned?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Is there any data motion or Broadcasts? Which table(s)?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | How could you reduce the number of tables being scanned?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | If you add a WHERE clause to get transactions for the month of May 2008 only, how does the explain plan change? This SQL can be found in the file lab5query1a.sql.  **AND transaction.transdate BETWEEN '2008-05-01' AND '2008-05-31'**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Execute another explain plan for a simple query without analyzing the data. This query is found in the file /home/gp/sql/load\_files/adv\_sql\_files/lab5query2.sql:  datamart=# **\i /home/gp/sql/load\_files/adv\_sql\_files/lab5query2.sql**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML39d62ebd.PNG |
|  | Which tables are being scanned?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Is there any data motion or Broadcasts? Which table(s)?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | If you partitioned your tables, how many “transaction” tables are being scanned? Why?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | **Summary**  You should execute an EXPLAIN statement with any new SQL that accesses large tables, any SQL that is running slowly, or against data objects that are new to the system. This helps you to determine, in advance of going into production, the likely path that the optimizer will take when performing the query. The EXPLAIN utility can produce a plan for any SQL statement, except EXPLAIN itself, so you are encouraged to take advantage of this functionality as often as possible. |

*End of Lab Exercise*

# Lab Improve Performance with Statistics

|  |  |
| --- | --- |
| **Purpose:** | In this lab you will gather statistics on all of the dimension and fact tables in your data warehouse. You will then do an EXPLAIN ANALYZE step for the two queries from the previous lab. |
|  | |
| **Tasks:** | Students perform the following task: Gather statistics and analyze queries. |
|  | |
| **References:** | Module 8 – Performance Analysis and Tuning   * Lesson: Improve Performance with Statistics |

Lab 20: Task – Gather Statistics and Analyze Queries

| **Step** | **Action** |
| --- | --- |
|  | Connect to the datamart database as gpadmin. |
|  | Analyze each of dimensions and facts tables using the ANALYZE command:   1. Analyze the dimensions.country table: datamart=# **analyze dimensions.country;** 2. Analyze the dimensions.customer table: datamart=# **analyze dimensions.customer;** 3. Analyze the dimensions.store table: datamart=# **analyze dimensions.store;** 4. Analyze the facts.transaction table datamart=# **analyze facts.transaction;**   C:\Users\cantot\AppData\Local\Temp\SNAGHTML39eec9d4.PNG |
|  | Execute the first query from the previous plan with the EXPLAIN command. This query is found in the /home/gp/sql/load\_files/adv\_sql\_files/lab5query1.sql file:  datamart=# **\i /home/gp/sql/load\_files/adv\_sql\_files/lab5query1.sql**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML39f01380.PNG  Are there any differences between the query plans with statistics?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Execute the second explain query plan from the previous lab using the EXPLAIN command. This query is found in the /home/gp/sql/load\_files/adv\_sql\_files/lab5query2.sql file:  datamart=# **\i /home/gp/sql/load\_files/adv\_sql\_files/lab5query2.sql**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML39f0a11f.PNG  Do you note any differences between this output and the output from the previous lab?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Execute these same queries with an EXPLAIN ANALYZE statement to compare actual execution demographics with the projected execution demographics. This query is found in the /home/gp/sql/load\_files/adv\_sql\_files/lab6query1.sql file:  datamart=# **\i /home/gp/sql/load\_files/adv\_sql\_files/lab6query1.sql** |
|  | C:\Users\cantot\AppData\Local\Temp\SNAGHTML39f2c64b.PNG |
|  | What do you note about this output when compared to an EXPLAIN?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | Execute the second query with EXPLAIN ANALYZE as well. This query is found in the /home/gp/sql/load\_files/adv\_sql\_files/lab6query2.sql file:  datamart=# **\i /home/gp/sql/load\_files/adv\_sql\_files/lab6query2.sql**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML39f52368.PNG |
|  | What do you note about this output when compared to an EXPLAIN?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | **Summary**  ANALYZE is a SQL command that updates the database statistics used by the query planner. In this use case, a query is run both before and after doing ANALYZE. After statistics are collected, the query planner chooses a HashAggregate operation over a much slower GroupAggregate operation. Without any statistics, the query planner could not estimate how many records might be returned, and therefore could not determine if there was sufficient work memory to do the aggregations in memory. The planner always takes the safe route and does aggregations by reading/writing from disk, which is significantly slower. |

*End of Lab Exercise*

This page intentionally left blank

# Lab Indexing Strategies

|  |  |
| --- | --- |
| **Purpose:** | In this lab, you determine requirements for secondary indexes on the fact table TRANSACTION. You will then create the secondary indexes. |
|  | |
| **Tasks:** | Students perform the following task: Create indexes to support queries |
|  | |
| **References:** | Module 8 – Performance Analysis and Tuning   * Lesson: Indexing Strategies |

Lab 21: Task – Create Indexes to Support Queries

| **Step** | **Action** |
| --- | --- |
|  | Here are some facts that you can use to develop your index strategies:   * + Store managers often query the transaction table for their store(s) using the storeid.   + Business Intelligence (BI) tools execute a lot of queries for stores and customers by state for one or more states.   + One of the executive dashboards queries transactions at stores based on the store feature grocery.   + Auditing directly accesses transactions by the transaction id.   Use the following matrix to determine which indexes you need to build to support the above analysis. A list of the tables that require indexes has been provided:   |  |  |  | | --- | --- | --- | | Table Name | Column(s) | Index Type | | transaction |  |  | | store |  |  | | customer |  |  | | store |  |  | | transaction |  |  | |
|  | Connect to the datamart database as gpadmin. |
|  | Using the CREATE INDEX statement syntax, create the above indexes for your database. Note that to index the transactions table, you will need to index each of the child tables.  The sample script, /home/gp/sql/load\_files/adv\_sql\_files/lab7createidx.sql, can be used for guidance.  Execute the query file:  datamart=# **\i /home/gp/sql/load\_files/adv\_sql\_files/lab7createidx.sql**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a030196.PNG |
|  | ANALYZE your database tables with new indexes:   1. Analyze the facts.transaction table: datamart=# **ANALYZE facts.transaction;** 2. Analyze the dimensions.store table: datamart=# **ANALYZE dimensions.store;** 3. Analyze the dimensions.customer table: datamart=# **ANALYZE dimensions.customer;**   C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a04f4ba.PNG |
|  | Determine if your indexes are being used by the optimizer by running the EXPLAIN utility with some simple queries that should use the index.  If the indexes are not being used, it is likely because of the small size of the data set. Try changing the enable\_seqscan configuration parameter to off if that is the case. Re-execute the EXPLAIN queries and then execute the queries. Be sure to set enable\_seqscan back to on before exiting psql. Execute an EXPLAIN against a query:  datamart=# **EXPLAIN**  **SELECT \* FROM dimensions.store ds, facts.transaction ft**  **where ds.storeid=ft.storeid;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a05668f.PNG |
|  | Disable sequential scans:  datamart=# **set enable\_seqscan=off;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a069656.PNG |
|  | Execute EXPLAIN against the query again:  datamart=# **EXPLAIN**  **SELECT \* FROM dimensions.store ds, facts.transaction ft**  **where ds.storeid=ft.storeid;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a072ae7.PNG |
|  | Set enable\_seqscan to on:  datamart=# **set enable\_seqscan=on;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a0791a5.PNG |
|  | Execute EXPLAIN against the query again:  datamart=# **EXPLAIN**  **SELECT \* FROM dimensions.store ds, facts.transaction ft**  **where ds.storeid=ft.storeid;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a081b4d.PNG |
|  | **Summary**  In most traditional databases, indexes can greatly improve data access times. However, in a distributed database such as Greenplum, indexes should be used more sparingly. The Greenplum Database is very fast at sequential scanning. Indexes use a random seek pattern to locate records on disk. Also, unlike a traditional database, the data is distributed across the segments. This means each segment scans a smaller portion of the overall data in order to get the result. If using table partitioning, the total data to scan may be even a fraction of that.  Greenplum recommends that you first try your query workload without adding any additional indexes. Indexes are more likely to improve performance for OLTP type workloads, where the query is returning a single record or a very small data set. Typically, a business intelligence (BI) query workload returns very large data sets, and thus does not make efficient use of indexes.  Note that the Greenplum Database will automatically create PRIMARY KEY indexes for tables with primary keys. If you are experiencing unsatisfactory performance, you may try adding indexes to see if performance improves. |

*End of Lab Exercise*

This page intentionally left blank

# Lab Advanced Reporting Using OLAP

|  |  |
| --- | --- |
| **Purpose:** | In this lab, you will be given a set of reporting requirements. You will write the SQL using the various OLAP functions discussed in the module. You may use the examples in the module as the basis for your queries. |
|  | |
| **Tasks:** | Students perform the following task: Create reporting queries using OLAP functions |
|  | |
| **References:** | Module 9 – Developing Reports Using Advanced SQL   * Lesson: Advanced Reporting Using OLAP |

Lab 22: Task – Advanced Reporting Using OLAP

| **Step** | **Action** |
| --- | --- |
|  | The reporting requirements for which you will develop queries using OLAP functions are as follows:   * Create a query that gives a rolling total of the Sales Amount by customer. Give the customer name. * Create a query that gives a monthly total of the Sales Amount for all stores with a grocery and a deli by month, by state summarizing at each control break (state and month). (Hint: try one of the grouping sets like rollup or cube.) * Create a query that lists the running total of each amount column by month, by store. Display the store name with the measures. (Hint: See the example in the module that uses a “derived table” or “in-line view”.) * Create a query that lists the top customer by Sales Amount, by month to support a customer rewards marketing campaign. (Hint: You will likely need 2 “derived tables” or *in-line views* to make this query work. It is admittedly tricky!) * Create a query that lists the stores, by month ranked by the most items sold.   **Note:** When working with dates, the easiest method to convert a date into month/year is with the following syntax: TO\_CHAR(date, 'yyyy-mm'). |
|  | Connect to the datamart database as gpadmin. |
|  | The query for the first requirement makes use of the SUM moving window function.  See the lab example in the, /home/gp/sql/load\_files/adv\_sql\_files/lab8query1.sql file if you are having problems writing this SQL. The content is as follows:  datamart=# **SELECT c.custname**  **,t.transdate**  **,t.SalesAmt**  **,SUM(t.SalesAmt) OVER (PARTITION BY t.customerid**  **ORDER BY t.transdate ASC**  **ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RollingTotalAmt**  **FROM facts.transaction t**  **INNER JOIN dimensions.customer c**  **ON c.customerid = t.customerid;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a0c91d4.PNG |
|  | The query for the second requirement works really well with the CUBE group function. You can use the ORDER BY clause of the SQL statement to display the results in a more readable format.  See the lab example in the /home/gp/sql/load\_files/adv\_sql\_files/lab8query2.sql file if you need assistance.  datamart=# **SELECT s.storename**  **,to\_char(t.transdate,'YYYY-MM') AS TransMonth**  **,SUM(t.SalesAmt) AS MonthlySalesAmt**  **FROM facts.transaction t**  **INNER JOIN dimensions.store s**  **ON s.storeid = t.storeid**  **WHERE s.grocery = true**  **AND s.deli = true**  **GROUP BY CUBE(s.storename,TransMonth)**  **ORDER BY 1,2;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a0d313d.PNG |
|  | The query for the third requirement is tricky.  Consider using a derived table or an in-line view to do a summary by month and store of each of the amount columns. Then apply the SUM window function to each of those columns in your table.  See the lab example in the /home/gp/sql/load\_files/adv\_sql\_files/lab8query3.sql file if you need a hint.  datamart=# **SELECT s.storename**  **,t.TransMonth**  **, SUM(t.salesamt) OVER (PARTITION BY s.storename ORDER BY t.TransMonth) AS TotalSalesAmt**  **, SUM(t.taxamt) OVER (PARTITION BY s.storename ORDER BY t.TransMonth) AS TotalTaxAmt**  **, SUM(t.discountamt) OVER (PARTITION BY s.storename ORDER BY t.TransMonth) AS TotalDiscountAmt**  **, SUM(t.couponamt) OVER (PARTITION BY s.storename ORDER BY t.TransMonth) AS TotalCouponAmt**  **, SUM(t.cashamt) OVER (PARTITION BY s.storename ORDER BY t.TransMonth) AS TotalCashAmt**  **, SUM(t.checkamt) OVER (PARTITION BY s.storename ORDER BY t.TransMonth) AS TotalCheckAmt**  **, SUM(t.ccamt) OVER (PARTITION BY s.storename ORDER BY t.TransMonth) AS TotalCCAmt**  **, SUM(t.debitamt) OVER (PARTITION BY s.storename ORDER BY t.TransMonth) AS TotalDebitAmt**  **, SUM(t.otheramt) OVER (PARTITION BY s.storename ORDER BY t.TransMonth) AS TotalOtherAmt**  **FROM (SELECT storeid**  **,TO\_CHAR(transdate,'YYYY-MM') AS TransMonth**  **, SUM(salesamt) AS SalesAmt**  **, SUM(taxamt) AS TaxAmt**  **, SUM(discountamt) AS DiscountAmt**  **, SUM(couponamt) AS CouponAmt**  **, SUM(cashamt) AS CashAmt**  **, SUM(checkamt) AS CheckAmt**  **, SUM(ccamt) AS CCAmt**  **, SUM(debitamt) AS DebitAmt**  **, SUM(otheramt) AS OtherAmt**  **FROM facts.transaction**  **GROUP BY 1,2) t**  **INNER JOIN dimensions.store s**  **ON s.storeid = t.storeid;** |
|  | C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a0dee93.PNG |
|  | The query for the fourth requirement is harder. You should consider a derived table or in-line view. However, you will need two:   * The first one does the summarization at your control breaks. * The second gets the ranking for you and then executes the final statement against the last derived table to get the results.   See the lab example in the /home/gp/sql/load\_files/adv\_sql\_files/lab8query4.sql file for details and hints. The contents of the file are displayed below:  datamart=# **SELECT Y.TransMonth**  **,Y.custname AS CustomerName**  **,Y.TotalSalesAmt**  **FROM (SELECT X.TransMonth**  **,c.custname**  **,X.TotalSalesAmt**  **,RANK() OVER (PARTITION BY X.TransMonth**  **ORDER BY X.TotalSalesAmt DESC) AS CustomerRanking**  **FROM (SELECT TO\_CHAR(t.transdate,'YYYY-MM') AS TransMonth**  **,t.customerid**  **,SUM(t.salesamt) AS TotalSalesAmt**  **FROM facts.transaction t**  **GROUP BY 1,2**  **) X**  **INNER JOIN dimensions.customer c**  **ON c.customerid = X.customerid**  **) Y**  **WHERE Y.CustomerRanking = 1**  **ORDER BY 1 ASC;** |
|  | C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a0f43a5.PNG |
|  | The query for the last requirement is similar to the query example in the lesson. Consider using a window partition to solve this query.  See the lab example in the /home/gp/sql/load\_files/adv\_sql\_files/lab8query5.sql file for hints. The content is as follows:  datamart=# **SELECT t.TransMonth**  **,s.StoreName**  **,t.TotalItemCnt**  **,RANK() OVER (PARTITION BY t.TransMonth**  **ORDER BY t.TotalItemCnt DESC) AS ranking**  **FROM (SELECT storeid**  **,TO\_CHAR(transdate,'YYYY-MM') AS TransMonth**  ,SUM(itemcnt) AS TotalItemCnt  **FROM transaction**  **GROUP BY 1,2 ) t**  **INNER JOIN store s ON s.storeid = t.storeid**  **ORDER BY 1 ASC;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a100b66.PNG |
|  | Connect to the faa database to execute queries against larger data sets.  datamart=# **\c faa** |
|  | The following query lists flights from Denver International airport to Logan International (Boston) airport:  faa=# **SELECT distinct carrierid, flightnum**  **FROM factontimeperformance f INNER JOIN dimairline al**  **ON f.airlineid = al.airlineid**  **INNER JOIN dimairport ap**  **ON f.originairportid = ap.airportid**  **WHERE f.originairportid in (**  **SELECT AIRPORTID**  **FROM dimairport**  **WHERE airportdescription = 'Denver, CO: Denver International')AND f.destairportid in (**  **SELECT AIRPORTID**  **FROM dimairport**  **WHERE airportdescription = 'Boston, MA: Logan International');**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a22f1de.PNG |
|  | List the number of flights for each airline carrier leaving New York (originwacid=22). Provide the information for each carrier per year. Display the grand total number of flights in the year.  Use the ROLLUP operator to display all the results at once:  faa=# **SELECT year, carrierID, count(\*)**  **FROM factontimeperformance**  **WHERE originwacid = 22**  **GROUP BY ROLLUP(year, carrierID)**  **ORDER BY year, carrierID;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a2f5add.PNG |
|  | List the number of flights leaving New York on a carrier and year basis. Display the total number of flights in the year and the total number of flights by a carrier. Use the cube operator to display all the results at once:  faa=# **SELECT year, carrierID, count(\*)**  **FROM factontimeperformance**  **WHERE originwacid = 22**  **GROUP BY CUBE(year, carrierID)**  **ORDER BY year, carrierID;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a307f2f.PNG |
|  | **Summary**  Use the OLAP grouping functionality whenever you need to display totals and sub-totals based on your group predicates.  Use the OLAP window expressions whenever you want to utilize the values in prior rows for comparison or further aggregation. This precludes having to make multiple scans of the same table in order to accomplish these comparisons. |

*End of Lab Exercise*

# Lab PostgreSQL Functions

|  |  |
| --- | --- |
| **Purpose:** | In this lab, you will create functions to perform simple, repeatable queries and tasks. You will create functions that use the SQL and the PL/pgSQL language. You will be required to create one user defined data type in order to correctly return data. |
|  | |
| **Tasks:** | Students perform the following tasks:   * + Create SQL functions   + Create PL/pgSQL functions |
|  | |
| **References:** | Module 9 – Developing Reports Using Advanced SQL   * Lesson: PostgreSQL Functions |

Lab 23: Task 1 – Create SQL Functions

| **Step** | **Action** |
| --- | --- |
|  | Connect to the datamart database as gpadmin. |
|  | Create an overloaded SQL function that returns a row or rows from the store dimension table with a parameter of either the storeid or the state. You can create the procedure in the public schema.  For hints on how to create the function, refer to the lab example in the /home/gp/sql/load\_files/adv\_sql\_files/lab10query1.sql file.  Remember that you want to return a SETOF to get more than one row.  datamart=# **CREATE OR REPLACE FUNCTION public.GetStore (whichstate CHAR(2))**  **RETURNS SETOF store AS $$**  **SELECT \***  **FROM dimensions.store**  **WHERE state=$1;**  **$$**  **LANGUAGE SQL;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a32f71c.PNG  datamart=# **CREATE OR REPLACE FUNCTION public.GetStore (whichid integer)**  **RETURNS SETOF store AS $$**  **SELECT \***  **FROM dimensions.store**  **WHERE storeid=$1;**  **$$**  **LANGUAGE SQL;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a335532.PNG |
|  | Test your function by selecting for the state of Missouri (MO) or any store id less than 30:  datamart=# **SELECT \* FROM public.GetStore('MO');** datamart=# **SELECT \* FROM public.GetStore(11);**C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a49cfde.PNG |
|  | Create an SQL function that takes as a parameter a table name (you can use any of the dimension tables) and returns the appropriate name column in uppercase. Call the function uppername. Create it in the public schema.  For hints on how to create the function, refer to the lab example in the /home/gp/sql/load\_files/adv\_sql\_files/lab10query2.sql file.  datamart=# **CREATE FUNCTION public.UpperName (store)**  **RETURNS text AS $$**  **SELECT UPPER($1.storename);**  **$$**  **LANGUAGE SQL;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a4a2dc6.PNG |
|  | Test the function by selecting the storeid**,** storename**,** public.Uppername(store)**,** city columns from store.  datamart=# **select storeid,storename,public.Uppername(store),city from store;**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a4a903f.PNG |

Lab 23: Task 2 – Create PL/pgSQL Functions

| **Step** | **Action** |
| --- | --- |
|  | Create a PL/pgSQL function to join the transaction fact table with the store and customer dimensions to return the transaction columns (transid, transdate, salesamt, taxamt, checkamt, cashamt, ccamt, debitamt), the store columns (storename, city, state) and the customer columns (custname, city, state).  Parameters are the starting and ending transaction dates. (HINT: The output will be a user defined data type.)  First, create the user defined data type and call it ReportTypeA.  datamart=# **CREATE TYPE ReportTypeA AS (**  **transid BIGINT,**  **transdate DATE,**  **salesamt NUMERIC,**  **taxamt NUMERIC,**  **checkamt NUMERIC,**  **cashamt NUMERIC,**  **ccamt NUMERIC,**  **debitamt NUMERIC,**  **storename CHARACTER VARYING,**  **storecity CHARACTER VARYING,**  **storestate CHARACTER VARYING,**  **custname CHARACTER VARYING,**  **custcity CHARACTER VARYING,**  **custstate CHARACTER VARYING);**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a4b0d3b.PNG |
|  | Create the function, RunReportA. Create the function in the public schema.  datamart=# **CREATE OR REPLACE FUNCTION public.RunReportA**  **(IN StartDate DATE,** **IN EndDate DATE)**  **RETURNS SETOF ReportTypeA AS**  **$BODY$**  **DECLARE r ReportTypeA%rowtype;**  **BEGIN**  **FOR r IN SELECT t.transid**  **,t.transdate**  **,t.salesamt**  **,t.taxamt**  **,t.checkamt**  **,t.cashamt**  **,t.ccamt**  **,t.debitamt**  **,s.storename**  **,s.city**  **,s.state**  **,c.custname**  **,c.city**  **,c.state**  **FROM facts.transaction t**  **INNER JOIN dimensions.store s**  **ON s.storeid = t.storeid**  **INNER JOIN dimensions.customer c**  **ON c.customerid = t.customerid**  **WHERE t.transdate BETWEEN StartDate AND EndDate LOOP**  **RETURN NEXT r;**  **-- return current row of SELECT**  **END LOOP;**  **RETURN;**  **END**  **$BODY$**  **LANGUAGE plpgsql;** |
|  | C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a4baec7.PNG |
|  | Test the function by running a report for the dates, 2008-07-01 to 2008-07-07:  datamart=# **select \*from public.runreporta ('2008-07-01'::date, '2008-07-07'::date);**  C:\Users\cantot\AppData\Local\Temp\SNAGHTML3a4c3506.PNG |
|  | **Summary**  You may find that using functions to generate reports is a simple way to store SQL in the database to ensure that any user executing the query cannot change it and the query will produce consistent results.  Functions provide a versatile way to perform complex transformation logic. Use functions in conjunction with temporary tables whenever you need to modify rows during ETL processing. Remember that you can also create temporary tables in the PL/pgSQL function!  Use dynamic functions whenever you need to create ad-hoc SQL. This will again keep consistent and repeatable SQL stored in the database. |

*End of Lab Exercise*

END OF LAB

This page intentionally left blank