

# Pivotal

A NEW PLATFORM FOR A NEW ERA

# Greenplum Architecture, Administration, and Implementation

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

2

## Welcome to the Greenplum Architecture, Administration, and Implementation.

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, eInput, E-Lab, EmailXaminer, EmailXtender, Enginity, 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.



## Copyright

Copyright © 2016 Pivotal Software, Inc. All rights reserved. This manual and its accompanying materials are protected by U.S. and international copyright and intellectual property laws.

Pivotal products are covered by one or more patents listed at <http://www.pivotal.io/patents>.

Pivotal is a registered trademark or trademark of Pivotal Software, Inc. in the United States and/or other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies. The training material is provided "as is," and all express or implied conditions, representations, and warranties, including any implied warranty of merchantability, fitness for a particular purpose or noninfringement, are disclaimed, even if Pivotal Software, Inc., has been advised of the possibility of such claims. This training material is designed to support an instructor-led training course and is intended to be used for reference purposes in conjunction with the instructor-led training course. The training material is not a standalone training tool. Use of the training material for self-study without class attendance is not recommended.

These materials and the computer programs to which it relates are the property of, and embody trade secrets and confidential information proprietary to, Pivotal Software, Inc., and may not be reproduced, copied, disclosed, transferred, adapted or modified without the express written approval of Pivotal Software, Inc.

Course Overview	
Description	The Greenplum Architecture, Administration, and Implementation course introduces students to the Greenplum environment, consisting of the Greenplum Database and supported systems. Students are introduced to fundamental concepts on data warehousing, business intelligence, and how Greenplum helps to solve business problems in managing and analyzing Big Data. Students will install, configure, and manage the Greenplum database system by evaluating logical models and business requirements to determine the best physical design for a Greenplum database.
Audience	The intended audience for this course includes Greenplum database implementers and designers, system administrators for the Greenplum environment, Greenplum database administrators, and database developers.
Prerequisites	To understand the content and successfully complete this course, a student must have an understanding of: •Basic UNIX or Linux commands •SQL Syntax •Fundamental relational database concepts
Pivotal	

© 2015 Pivotal Software, Inc. All rights reserved.

4

## Course Overview

The Greenplum Architecture, Administration, and Implementation course provides you with the foundation you need to interact with, manage, and develop for your Greenplum environment, as an administrator, implementer, or developer. You will gain insight into the benefits that Greenplum offers to your organization in managing and analyzing your data.

Once you install and configure Greenplum, you will manage the data by evaluating logical models and business requirements to determine the best physical design for your database. You will consider specific performance impacts to retrieving your data based on how your data is stored within the database.

To successfully complete this course, you should have:

- A working knowledge of basic UNIX or Linux commands to help you navigate the Linux environment
- Some basic understanding of SQL and SQL syntax
- Fundamental relational database concepts

## Course Objectives

Upon completion of this course, you should be able to:

- Gain a basic understanding of data warehousing concepts and be able to describe PostgreSQL to develop a foundation for understanding the Greenplum solution.
- Learn about the Greenplum architecture and hardware solutions to support the architecture so that they understand how to properly implement a solution based on their company's business needs.
- Build and implement a Greenplum software solution to manage the Greenplum environment and database through a variety of Greenplum utility tools and PSQL.
- Learn PostgreSQL and Greenplum specific SQL features and functions to manage data and optimize SQL query performance in a Greenplum database.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

5

## Course Objectives

On completion of the course, you should be able to:

- Identify and explain data warehousing concepts and define what PostgreSQL is as well as its relationship to Greenplum
- Identify the major components within the Greenplum architecture and highlight the importance of certain hardware solutions that support the architecture
- Build and implement a Greenplum software solution in to manage the Greenplum database, database instances, and data using a variety of Greenplum utility tools and the PSQL client
- Use both PostgreSQL and Greenplum-specific SQL commands and functions to manage data and optimize SQL query performance in a Greenplum database

# Agenda

	Modules/Lessons	Labs
Day 1	<ul style="list-style-type: none"><li>• Greenplum Fundamental Concepts<ul style="list-style-type: none"><li>• The Basics of Data Warehousing</li><li>• Greenplum Concepts, Features, and Benefits</li><li>• Greenplum Architecture – Shared Nothing and MPP</li><li>• Greenplum Product Overview</li></ul></li><li>• Database Installation and Initialization<ul style="list-style-type: none"><li>• Systems Preparation and Verification</li><li>• Greenplum Database Initialization</li></ul></li><li>• Greenplum Database Tools, Utilities, and Internals<ul style="list-style-type: none"><li>• Using PSQL Client and Greenplum Utilities</li></ul></li></ul>	<ul style="list-style-type: none"><li>• Lab Preparation and Initialization</li><li>• Greenplum Product Overview</li></ul> <ul style="list-style-type: none"><li>• Systems Preparation and Verification</li><li>• Pivotal Greenplum Database Initialization</li></ul> <ul style="list-style-type: none"><li>• Using the PSQL Client and Greenplum Utilities</li></ul>
Day 2	<ul style="list-style-type: none"><li>• Greenplum Database Tools, Utilities, and Internals (Cont)<ul style="list-style-type: none"><li>• Pivotal Greenplum Command Center</li><li>• Greenplum Database Server Configuration</li><li>• Greenplum Database Internals</li></ul></li><li>• Defining and Securing the User Database<ul style="list-style-type: none"><li>• Data Definition Language</li><li>• Data Manipulation Language and Data Query Language</li></ul></li></ul>	<ul style="list-style-type: none"><li>• Greenplum Database Server Configuration</li><li>• Database Internals</li></ul> <ul style="list-style-type: none"><li>• Data Definition Language</li><li>• Data Manipulation Language and Data Query Language</li></ul>

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

6

## Agenda

The course is a mixture of lecture and lab over a period of five days. The course is divided into eight modules which define specific high-level related concepts and activities for implementing and managing Greenplum. These modules are:

- **Greenplum Fundamental Concepts** – You will explore the concepts of data warehousing and how these concepts apply to Greenplum. You will then examine the features and benefits of implementing Greenplum in your environment. Finally, you will examine the concepts of shared nothing and MPP to understand how they help Greenplum to scale as well as it does.
- **Database Installation and Initialization** – A solid hardware solution greatly improves your Greenplum queries. You will take an in-depth look at the Greenplum product itself and examine these hardware solutions. You will prepare a system for Greenplum installation, perform the installation, and verify that all requirements are met before proceeding with initializing your Greenplum database. A variety of tools will allow you to connect to and manage the Greenplum environment. Once you learn how to use these tools, you will manage your Greenplum database instance.
- **Greenplum Database Tools, Utilities, and Internals** – Commonly used tools and utilities are examined in this module. In this module, you will examine and use utilities that provide access to the database and configure, monitor, and manage various aspects of the database.
- **Defining and Securing the User Database** – On installing the database, you must have a solid understanding of the database objects you will manage and the syntax available for managing these objects and your data. You will manage workload resources to protect not only data, but also the resources within the database environment. To do that, you will secure the environment by creating appropriate privileges and roles.

# Agenda

	Modules/Lessons	Labs
Day 3	<ul style="list-style-type: none"><li>• Defining and Securing the User Database (Cont)<ul style="list-style-type: none"><li>▪ Roles, Privileges, and Resources</li></ul></li><li>• Data Loading and Distribution<ul style="list-style-type: none"><li>▪ Implementing Table Storage Models, Compression, and Tablespace</li><li>▪ Data Loading</li><li>▪ Table Partitioning</li></ul></li><li>• Database Management and Archiving<ul style="list-style-type: none"><li>▪ Managing the Greenplum Database</li><li>▪ Backups and Restores</li></ul></li></ul>	<ul style="list-style-type: none"><li>• Roles, Privileges, and Resources</li><li>• Controlling Access</li><li>• Table Management</li><li>• Data Loading</li><li>• Table Partitioning</li></ul> <ul style="list-style-type: none"><li>• Managing the Greenplum Database</li><li>• Backups and Restores</li></ul>
Day 4	<ul style="list-style-type: none"><li>• Data Modeling and Design<ul style="list-style-type: none"><li>▪ Data Modeling</li><li>▪ Physical Design Decisions</li></ul></li><li>• Performance Analysis and Tuning<ul style="list-style-type: none"><li>▪ JOIN Tables – Types and Methods</li><li>▪ Database Tuning</li><li>▪ Query Profiling</li><li>▪ Explain the Explain Plan – Analyzing Queries</li></ul></li></ul>	<ul style="list-style-type: none"><li>• Data Modeling</li><li>• Physical Design Decisions</li></ul> <ul style="list-style-type: none"><li>• Database Tuning</li><li>• Explain the Explain Plan – Analyzing Queries</li></ul>

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

7

## Agenda (continued)

- **Data Loading and Distribution** – Once the database has been created, you will load the data. You must make decisions on how the data will be distributed. You will therefore need to understand what table partitioning is and how it can aide you.
- **Database Management and Archive** – You will learn commands that will aid you in managing the database instance as well as creating data backups and restoring the data as necessary.
- **Data Modeling and Design** – While earlier modules examined data loading and some data distribution techniques, you will more closely examine how to model your data and make key physical design decisions which will impact how the data is distributed in your environment.
- **Performance Analysis and Tuning** – Another in-depth subject, this module takes a closer look at specific commands and methods that can impact how your data is retrieved. You look at methods of examining your queries so that you develop appropriate queries to maximize performance gained based on how your data is distributed. You examine how to improve statistical analysis of your data to improve the overall query performance. You also examine whether or not to index your tables, implement OLAP methods in your queries, create PostgreSQL functions, and apply specific performance tuning tips for your data and your queries.

## Agenda

	Modules/Lessons	Labs
Day 5	<ul style="list-style-type: none"><li>• Performance Analysis and Tuning (Cont)<ul style="list-style-type: none"><li>• Improve Performance with Statistics</li><li>• Indexing Strategies</li></ul></li><li>• Developing Reports Using Advanced SQL<ul style="list-style-type: none"><li>• Advanced Reporting Using OLAP</li><li>• PostgreSQL Functions</li><li>• Advanced SQL Topics and Performance Tips</li></ul></li></ul>	<ul style="list-style-type: none"><li>• Improve Performance with Statistics</li><li>• Indexing Strategies</li><li>• Advanced Reporting Using OLAP</li><li>• PostgreSQL Functions</li></ul>

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

### Agenda (continued)

- **Developing Reports Using Advanced SQL** – Analysts use a combination of custom ad-hoc queries and Business intelligence tools to generate reports. The module highlights the various OLAP methods available to generate reports as well as how to build functions to access commonly used methods.

This slide is intentionally left blank.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.



# Module 1: Greenplum Fundamental Concepts

This module sets the foundation for implementing a Greenplum solution in your environment.

Upon completion of this module, you should be able to:

- Identify the basic elements and common design methodologies of data warehousing
- Describe the features and benefits of implementing Greenplum
- Describe the Greenplum architecture in terms of shared nothing and the Massively Parallel Processing (MPP) design
- Identify and describe the components of the Greenplum architecture and describe how Greenplum supports redundancy and high availability

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

1

Greenplum is a solution built to support the next generation of data warehousing and large-scale analytics processing. In this module, you will learn foundation concepts that will be applied when implementing the Greenplum solution both during this course and in your environment. You will:

- Examine the basic elements of a data warehouse and identify common design methodologies used when implementing a data warehouse.
- List and describe the features available from Greenplum and the benefits of implementing Greenplum in an environment.
- Examine a high-level overview of the shared-nothing massively parallel processing (MPP) design.
- Identify and describe the components of the Greenplum architecture and describe how Greenplum supports redundancy and high availability.

# Module 1: Greenplum Fundamental Concepts

## Lesson 1: The Basics of Data Warehousing

In this lesson, you review the concepts of a data warehouse and identify the basic elements that make up a data warehouse.

Upon completion of this lesson, you should be able to:

- Describe the Greenplum database
- Define the terms Big Data and data warehouse
- Differentiate between OLTP and OLAP systems
- List the basic elements of a data warehouse
- Highlight the role that ETL and ELT plays in data warehousing
- Identify commonly used methodologies in a data warehouse

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

2

The lesson introduces you to the concept of data warehousing as well as the basic elements and methodologies used in building and implementing a data warehouse. You examine concepts meant to differentiate between online transaction processing systems and online analytic processing systems. Additionally, you examine the role that extract, transform, and load (ETL) and extract, load, and transform (ELT) play in creating your data warehouse.

## Greenplum – Database of Choice for Deep Analytics

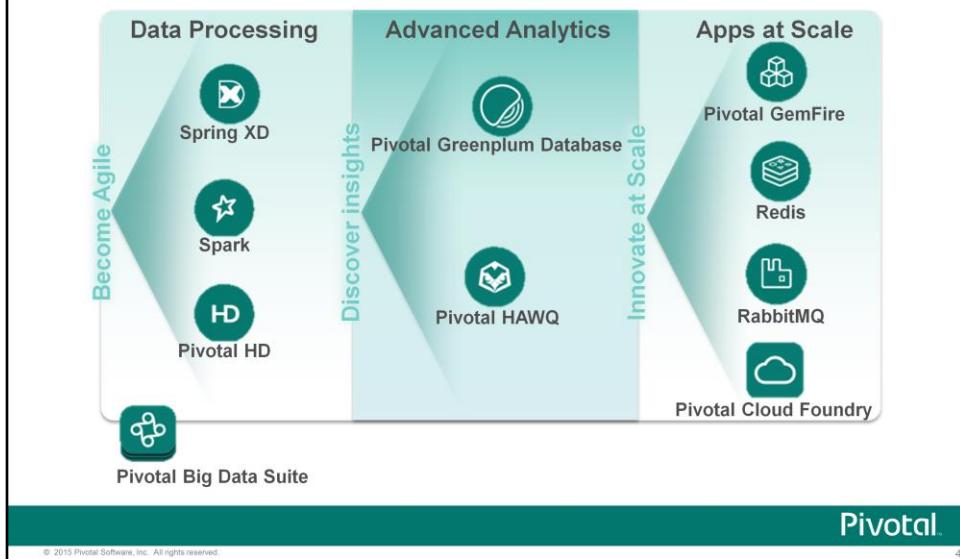


Greenplum is the database of choice for deep analytics. Pivotal Greenplum Database, designed for business intelligence and analytical processing, adds value in that you can now ask questions you were never able to ask before and in return, obtain valid and solid answers.

Greenplum Database is built to support the next generation of Big Data warehousing and large-scale analytics processing. It stores and analyzes terabytes to petabytes of data. The Greenplum Database was conceived, designed, and engineered to allow customers to take full advantage of large clusters of increasingly powerful servers, storage, and internet switches.

The number of companies looking for a solution to tackle the growth and management of data in their environment is ever increasing.

## Pivotal Greenplum Database in the Pivotal Big Data Suite Stack



Greenplum Database has evolved to become a backbone product for advanced analytics of the Pivotal Big Data Suite. The core principles behind the Big Data Suite allow an organization to:

- Become agile where they are able to more quickly and manage assets in the business data lake. The ability to take disparate sources of data and provide a centralized access point to that data is becoming a necessity in today's move towards growth and accessibility.
- Discover more insights by employing predictive analysis against large amounts of data. The combination of an enterprise ad-hoc solution and analytical data warehouses offer flexibility for gaining insight into large datasets that would otherwise be difficult to parse.
- Innovate at scale by acting on data to create customized experiences for different customers, including deploying mobile applications from a single framework to multiple sources with context-aware features based on real-time, in-memory data stores and reliable message queues.

With the availability and consolidation of all these toolsets, Greenplum Database, introduced with cloud storage as part of the Big Data solution has now moved to provide strong support in the enterprise data lake.

Greenplum can be physical or virtual, can be executed on any type of hardware, and can provide the flexibility customers are looking for.

Scalability, maintainability, and real-time provisioning in a virtual environment has great appeal as the industry moves towards enterprise data cloud.

Before continuing to discuss the Greenplum database, let us fully explore the concept of Big Data and the data warehouse.

## What Is Big Data?

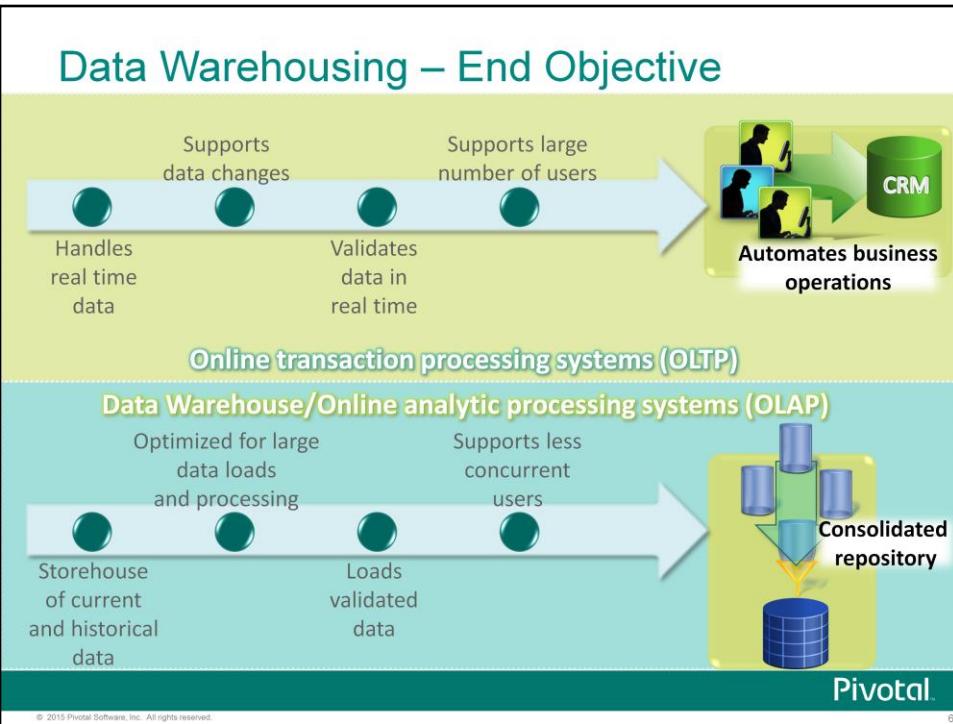


*Big Data* refers to the tools, processes, and procedures used to create, manipulate, and manage very large data sets, on the order of terabytes and petabytes of data.

Big data is changing and Greenplum continues to lead in changing how it is perceived. In yesterday's data warehouse and analytic infrastructure, big data is proprietary, expensive, monolithic, loaded once a, and ultimately, slow. The demands of data and business are changing.

Time to business value is now measured in minutes. Data may need to be loaded every 60 seconds and support must be there to analyze data in smaller time frames. This helps a company spot trends that are occurring in real-time. This change represents the move to the next generation of Big Data, where queries can be performed in real time against petabytes of data.

The proliferation of data presents a challenge in traditional databases, data warehousing, business intelligence, and analytics. Real-time processing represents the merging of OLTP and DW. The data warehouse has to be more agile to support real-time analysis.



To understand the end-goal of a data warehouse, it is important to distinguish data warehouses from transactional processing systems.

Online transaction processing systems, or OLTP systems, automate day-to-day business operations and processes by handling data in real time and allowing modification of existing data, removal of data, and addition of new data. All data is validated in real time. These systems normally support thousands of concurrent users, handling very fast and simple ad-hoc queries.

A data warehouse, also referred to as an online analytic processing system, is a consolidated repository designed for reporting and analysis. Storing current and historical data, it is optimized for bulk loads and processing of large amounts of data. These systems often rely on bulk loads to load current data into the system. Years of data may be maintained within the database, allowing long term analysis of data. Lower number of concurrent users perform more complex queries that may take minutes or hours to complete.

## OLTP (Transactional Database System) versus OLAP (Data Warehouse)

OLTP Systems (Transactional Database Systems)	OLAP Systems (Data Warehouse)
Supports day-to-day business operations	Supports reporting and analytics
Highly normalized (redundancies are reduced)	Highly denormalized (the same data may be replicated in multiple tables)
Larger number of tables with less data per table	Smaller number of tables with large amounts of data
Optimized for write operations (inserts, updates, and deletes)	Optimized for read operations (not necessarily meant for updates and deletes)
Users often send simple queries to return or update small number of rows to the requestor	Complex queries involving aggregations are normally used to return a large number of rows
Referential integrity is enforced	Data can be redundant and referential integrity is not enforced

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

7

Businesses use a combination of these complementing technologies to support business processes and analytic processing. On-line transaction processing systems are used to support day-to-day business processes such as would be found in sales transactions and invoicing systems, ATM or other daily banking transactions, or tracking for logistics and shipping systems, just to name a few. The data in OLTP systems is highly normalized, reducing redundancies, increasing the number of tables, and normally creating a logical grouping of tables. This data normalization improves data integrity and reduces the chance of anomalies. This type of environment is optimized for write operations where you experience frequent data changes affected by inserts, updates, and deletes. OLTP systems are also well tuned for common read queries that return a low number of data records.

Online analytic processing systems are designed to handle complex queries that may involve multiple aggregations. Data in these systems is denormalized with fewer tables, making it more attractive for read operations. Data may be refreshed but is normally not updated. The focus of this data is to provide line of business users and executives the information they need to help drive business decisions by looking at historical and temporal data.

## Expected Outcomes: Transactions Versus Reporting

What is the total amount Tom Franklin has across his accounts?

ATM Request:  
Display account balance for account 201.

Transactions to and from operational systems

What are the last five deposits on Sarah Macarthy's checking accounts?

What is the total non-interest income across the northeast states for the last 5 years?

Which customers have maintained an active account for longer than three years over the past 10 years?

Report the new account volume by geographic region and type for the past 3 years

Reports generated from data warehouses

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

8

Transactional or operational systems deal with day-to-day queries that operate the business. Users of operational systems initiate transactions, take orders, invoice customers, ship orders, or book events. What they ask of, or add to transactional systems are usually small (on the order of a single transaction at a time) and require fast response times.

Data warehouses are historical tracking systems that allow you to keep a bevy of information over the course of years instead of on a daily basis. In a banking scenario, it may be keeping track of the daily interest rate on an account over a period of years, whereas a transactional system keeps track of the current interest rate. This allows a consumer of this system to track trends over large periods of time. It also means that data in this infrastructure is not only redundant, in that it is repeated in different tables, but you may find multiple iterations of data as it is applied to a single entity.

Continuing with the interest rate example, the transactional system will show the current interest rate as it applies to an individual's account. In a data warehouse, that same customer may show up 100 times in the table because the interest rate on their account has changed over the past 3 years. The ability to track this type of information may help identify trends over the long term in customers' banking behaviors.

## What Is a Data Warehouse?

A **Data Warehouse** is the central repository of information gathered about the Enterprise that is required to support **Business Decision Making**.



A data warehouse is a culmination of information gathered about the enterprise. The information, which can consist of a variety of topics including sales data, organizational data, operational data, and inventory data, is used to support business decision making.

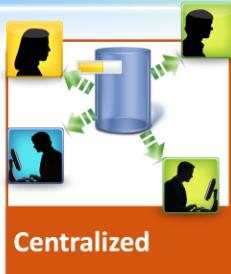
The data warehouse acts as a single source of truth, often times collecting data from multiple stores, making it easier to retrieve and analyze the data from a centralized collection.

In a data warehouse, data must be:

- **Centralized** – Data, while it may be accessible from multiple locations or other data marts, should be centralized when in a data warehouse
- **Agreed upon** – If the data is to be a single source of truth, it must be agreed upon. Data must be clearly defined so that it is useful when analyzed. This “agreed upon” terminology refers to the metadata that will be used across all business functions.
- **Easily accessible** – Data should be clearly labeled and consistent. Inconsistent data yields inconsistent reports which can ultimately affect business decisions.
- **Timely and relevant** – Data can give a company the competitive edge, if it is readily available and relevant to business needs. You should therefore be able to extract data as quickly as possible to help a business not only meet its goals, but also to gain a competitive edge in their market.

## Data Warehouse – the Centralized Repository

The Data Warehouse must contain a **single view of the business, or the Single Version of the Truth**, in order to be valuable to the enterprise.



- Conflicting answers are possible if a company has multiple databases or data marts
- A centralized data warehouse should be the platform to unify all the versions of “the truth”, coming from the company’s transactional systems
- Lack of a well-defined “Single Version of the Truth” can lead to a distrust of information

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

10

A centralized repository presents a single view of the data. If you do not have a centralized repository, this creates problems for aggregating and analyzing your data, including creating conflicting answers to business questions.

This centralized repository should act as the single version of truth within the enterprise. This eliminates the need to validate the data by accessing other databases or data marts within the enterprise. The organization must be able to trust that the information contained within the data warehouse is valid and therefore, the single version of truth for the enterprise.

## Data Warehouse – the Data Must Be Agreed Upon

*The components of the Data Warehouse, subject areas, and the data elements should be **clearly defined and agreed upon** by the Business!*



- Report metrics should have common and consistent definitions and business logic
- Achieving agreement among stakeholders that provide and extract data from the Data Warehouse is very important and often difficult task for Cross Functional Analysts

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

11

Data that is not clearly defined and agreed upon may result in inaccurate or misleading reports.

Consider the following scenario: A sales report indicates the number of servers sold by field sales in the last quarter. For this report, sales figures include initiated sales contracts, or when the order has been placed. The sales organization within the company uses this business logic to determine their sales figures. Finance however relies on a paid invoice to determine that a sale has been. A single report relying on the first business logic yields skewed sales figures for organizations that determine quarterly sales figures based on the second logic.

Data must therefore be agreed upon to be useful to the entire business as a whole. This requires clear definition in business logic and rules.

## Data Warehouse – Data Must Be Easily Accessible

*The contents of the Data Warehouse should be **clear**, **easy to navigate**, and access should be **characterized by defined response time**.*



- Data elements must be clearly and correctly labeled
- Data definitions must be consistent and easy to find
- Data must be available per the service level agreement
- Confidential data must be protected
- Data integrity must be enforced at all levels

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

12

Collecting and analyzing data relies on data that is:

- Clearly defined and correctly labeled
- Consistent and easy to find
- Available at the defined performance level
- Confidential
- Valid, where data integrity is maintained at all levels

Invalid, incorrect, or mislabeled data makes collection and analysis a difficult task. To extract data, an analyst must know precisely what data to look for and expects that the data is valid at the time of collection. To protect privacy, confidential data must be protected from unauthorized access.

## Data Warehouse – Data Must Be Timely and Relevant

*Data Warehouse must be refreshed as often as required for relevant reporting and analysis to support timely decision making!*



- The data warehouse must have the *right* data to support decision making
- Stale data translates to late or poor decisions
- The only true output from a data warehouse consists of the business decisions made after reviewing the information retrieved from the data warehouse

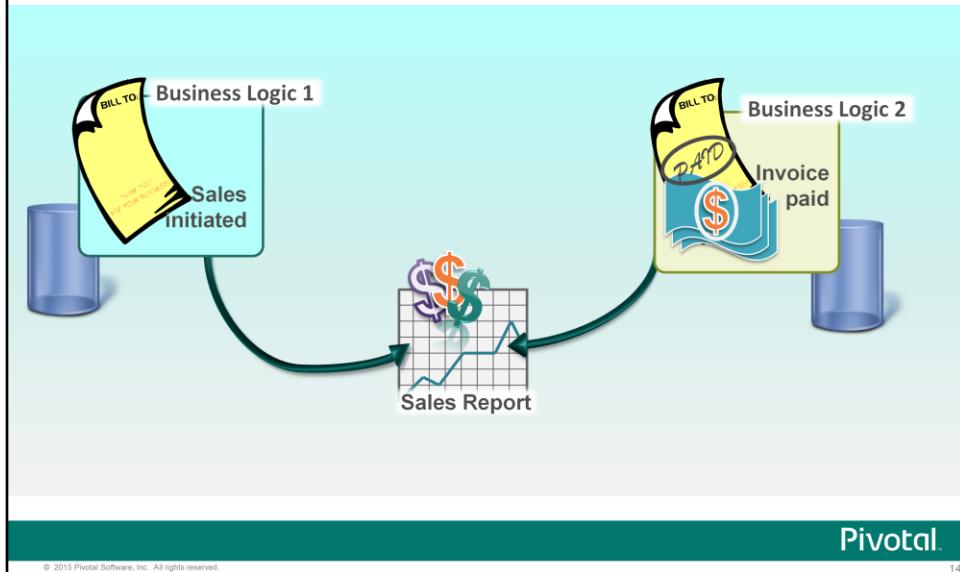
Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

13

A financial institution needs to analyze trading volumes on the NASDAQ. For this information to have any relevance, freshly acquired data must be available every five minutes. The ability to quickly analyze new data allows an organization to make better business decisions.

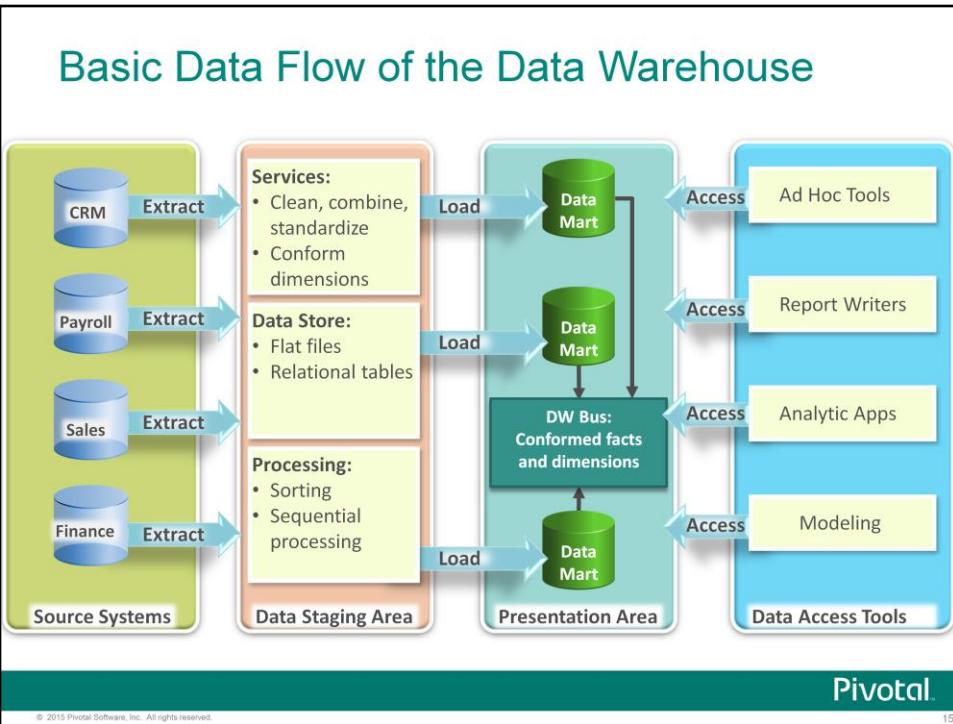
## Data Warehouse Analysis Example – Sales Reporting



Here is a simple case where there are two business rules being applied to data to generate a single report for an organization or company. One system looks at sales from the point where sale orders were generated. A second system looks at sales from the point at which the sales order has been completed and the invoice has been paid. The second system will be more tightly coupled with inventory and billings than the first system. Both systems use the term sales to indicate the end result of the business logic applied.

However, a report generated takes the term, or field, sales, and generates a report to show how the organization as a whole is doing. Because the two systems apply different logic to the sales terminology, the report does not necessarily reflect what the organization is looking for.

By applying basic data warehousing concepts that examines the business models, actors, business logic, and other requirements, the potential for inaccurate reporting and analysis can be greatly reduced.

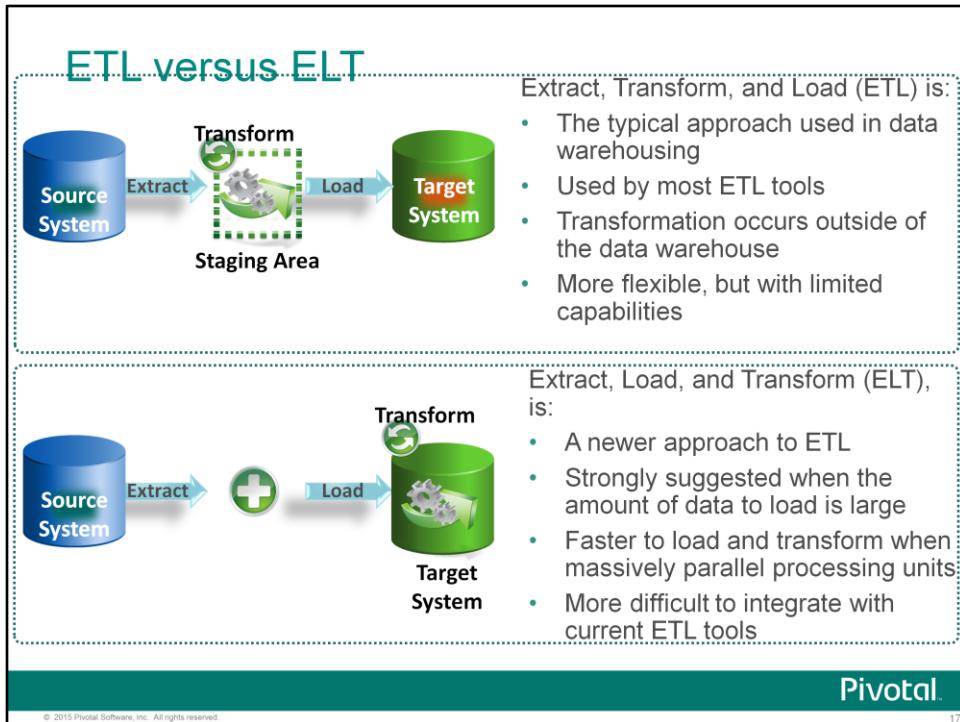


The basic components or elements of a data warehouse include:

- **Source System** – An operational system of records which function is to capture the transactions of the business. There is usually more than one.
- **Data Staging Area** – A storage area and set of processes that cleanses, combines, removes duplication and prepares the data for insertion into the Data Warehouse. The staging area may be spread over a number of systems. It does not provide query and presentation services to the user.
- **Operational Data Store (ODS)** – A subject-oriented, integrated, current and detailed-only collection of data in support of an organization's need for up-to-the-second operational reporting. Could be used as a staging area for Data Warehouse.
- **Data Mart** – In the presentation area, you have the Data Mart, a dependent, or independent area of data storage, usually related to one subject area. An Independent Data Mart stands on its own and bears no relation to any other subject area in the Data Warehouse. A Dependent Data Mart has relationships defined outside of its own subject area.
- **Business Intelligence** – A set of tools and business logic in the data access tools area that turns the data from the Data Warehouse into useful information for Decision Support.

## **Basic Data Flow of the Data Warehouse (Continued)**

- **Metadata** – All of the information in the Data Warehouse Environment that is not the data itself. Often called the “Data about the Data”.
- ETL – a set of processes in the Data Warehouse system to:
  - Extract the data from external systems
  - Transform it according to the business logic
  - Load it into target DW tables



In ETL, or extract, transform, and load, you extract the data from a source system, perform the transformation of the data on the ETL or staging system, and load the data to the target or presentation area, which would be the data warehouse. While the solution is flexible, it does have several limitations. One major factor to consider when designing a solution using ETL is that the solution is tied to hardware to perform the transformation. The source data is moved from one or more source systems or data marts and must be cleaned, verified, and transformed based on the data format of the output, before moving to the warehouse. The industry has multiple tools that support the use of ELT in the data warehousing environment.

In ELT, or extract, load, and transform, you extract the data from the source system and perform the load and transform in the target system or presentation area. The staging area can be defined within the target system itself, allowing you to utilize existing hardware. This method is strongly advised when working with large amounts of data. While it is powerful, it is difficult to integrate with current tools such as Informatica or Data Stage. The toolset for this methodology is limited. However, this method allows you to take advantage of the powerful featureset offered by Greenplum, including the massively parallel processing units which allow you to perform the load and transformation much faster.

## Data Warehouse Methodologies

Two commonly used methods to model a data warehouse are:

- Dimensional model – Divides transactions into facts and dimensions
- Normalized model – Stores data in subject divided areas or tables (level of normalization is not as high as seen in OLTP systems)

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

18

There are essentially two commonly used methods to model a Data Warehouse. This section will discuss each in detail. The two methods are:

- **Dimensional model** – Divides transactional data into facts and dimensions. Facts usually stand for various measures and dimensions give a context information about the facts. This is the most common data warehouse model.
- **Normalized model** – Stores data in subject divided areas (tables) and derives its name from the Codd data normalization methods. The level of normalization for this approach is usually not as high as for the typical OLTP data models.

## Dimensional Model

The dimensional model:

- Contains the same information as the normalized model
- Stores data in a symmetric form using the following table types:
  - Facts – Usually the largest table that is meant to store measurements data (metrics) about the business operations.
  - Dimension - One of the companion sets of tables describing the metrics data from the fact tables.
- Has the following positive characteristics:
  - Is easy to use and understand
  - Usually provides better query performance
- Has the following negative characteristics:
  - Requires complicated data loads to maintain integrity
  - Difficult to modify

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

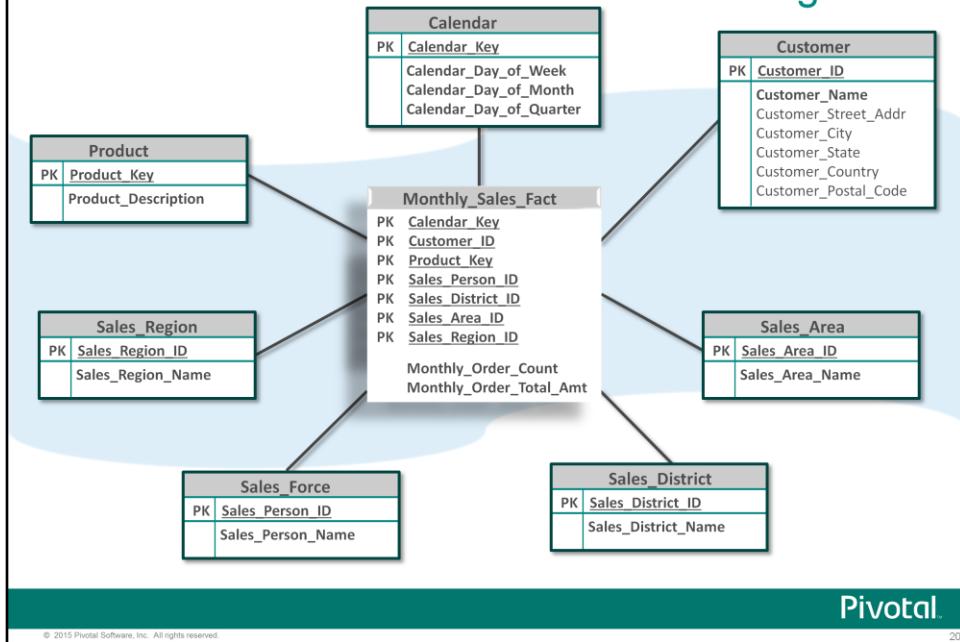
19

The dimensional model contains the same information that is supposed in the normalized model. It differs in that it stores data in a symmetric form using fact and dimension tables.

A fact table is usually the largest table and is meant to store measurement data, or metrics, about the business operations. The dimension table describes the metric data from the fact table. Each fact table row holds a foreign key ID that points to a row in a dimension table.

The dimension table is easier to understand and use and overall, performs better on query operations. That ease of use comes at a cost. It requires complicated data loads to maintain data integrity. As such, it can be difficult to modify once established. This makes it difficult to maintain the database in the face of business changes.

## Dimensional Model – Star Schema Diagram



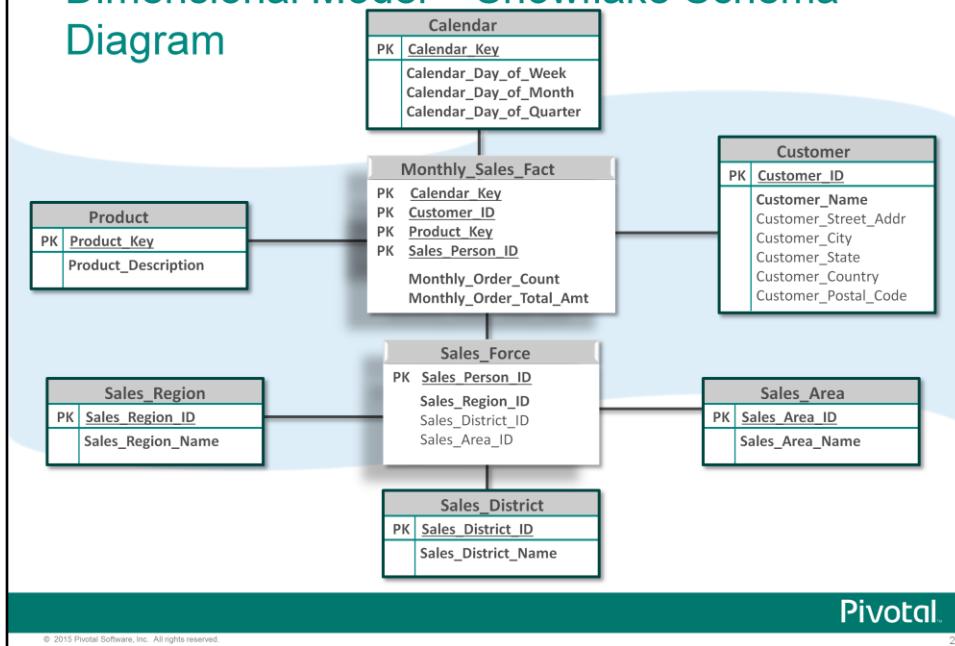
20

We will briefly examine two types of dimensional models that can be implemented: the star and snowflake schemas.

In the star schema, the fact table, **Monthly\_Sales\_Fact**, holds the main data. The remaining tables are dimension tables which describe the facts presented in the fact table. The fact table contains a compound primary key which are actually foreign keys from the dimension tables. The dimension tables are simpler, consolidating data in the most granular column, and are therefore rendered in second-normal form.

The fact table is usually in third normal form because all of the data depends on either one dimension or all of them, not a combination of just a few dimensions.

## Dimensional Model – Snowflake Schema Diagram



21

In the snowflake schema, multiple centralized fact tables connect to multiple dimension tables. The arrangement in the entity relationship diagram is such that the schema is represented in the form of a snowflake. With the snowflake schema, dimensions are normalized into multiple related tables. A single dimension table may therefore have multiple parent tables.

In this example, the fact table only has four primary keys, one for each table to which it is directly connected. A dimension table, **Sales\_Force**, is directly associated with other dimension tables.

## Normalized Model

The normalized model:

- Requires all data elements in each row must rely on only the primary key of that table
- Consists of:
  - Entity – A specific set of data uniquely identified by primary key and stored in a single table
  - Relationship – Describes how two or more entities are related
- Has the following positive characteristics:
  - Simple for data loading
  - Easier to maintain data integrity
  - Slower performance due to multiple table joins
- Has the following negative characteristics:
  - Difficult to understand and use
  - Complicated for query writing

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

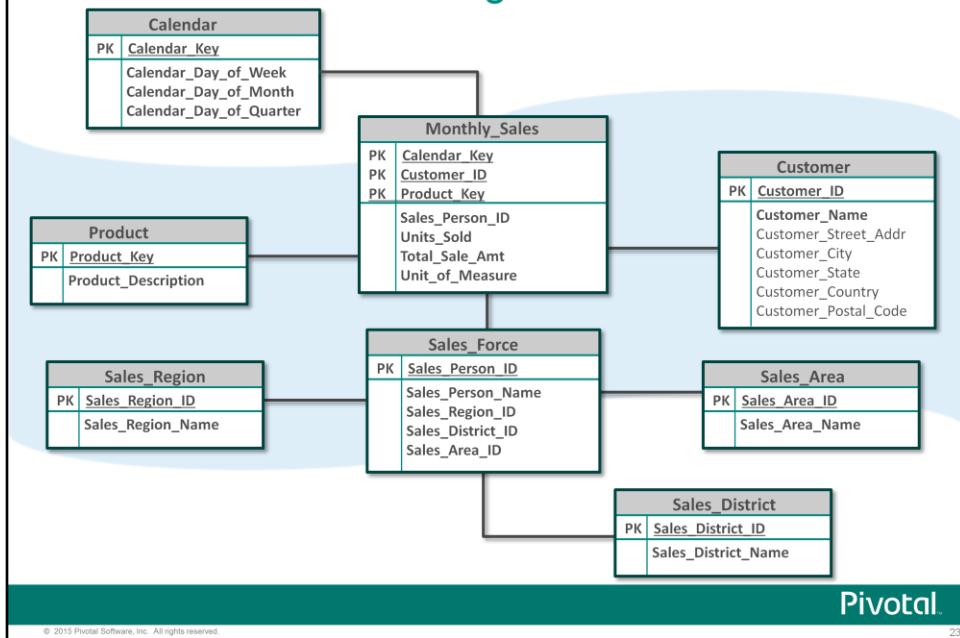
22

The normalized model, also known as the third normal form, is based on the normalization rules defined by E. F. Codd. It maintains that all of the data elements in each row must rely on the primary key and only on the primary key of that table. Tables are grouped together by *subject areas* that reflect general data categories.

The main components of the normalized model are the entity and relationship. The entity is a specific set of data uniquely identified by the primary key and stored in a single table. The relationship describes how two or more entities are related to each other. For example, it may describe how a customer is related to a product.

The normalized model is simple for data loading and easier to maintain for data integrity. On the negative side however, it is difficult to use and understand. It can be complicated when developing queries, and can be hampered by performance issues due to the need to join multiple tables together to gather the information requested by the query.

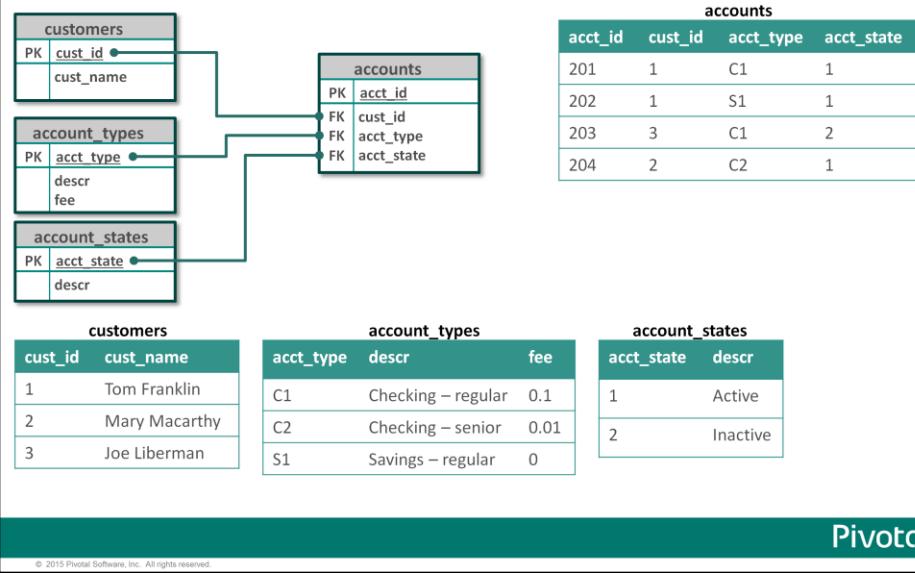
## Normalized Model Diagram



23

The diagram reflects the normalized model as you would find in an OLTP system, not in a data warehouse environment. Each entity is defined within its own unique table. Though the sales table has a compound primary key that consists of multiple foreign keys, sales is itself an entity and is described by those primary keys defined. It is easier to extend this diagram if you need to add additional entities – you would define separate table. However, you can see how quickly performance may decrease when you require multiple tables to be joined together to respond to a query.

## Data Modeling Example: Normalized Data in an OLTP System



Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

24

The example shown highlights normalized data, where tables are subject-oriented. Customer information is kept separately from account information, with additional descriptor tables available to provide definition for specific columns in these tables.

Highly normalized data eliminates redundancies, requiring multiple joins to access specific information. While this model is ideal for transactional systems, it introduces long response times when used to generate reports. Data warehouses are used to generate large reports based on millions of rows. Joining tables together where multiple tables are large introduces long processing time.

## Data Modeling Example: Normalized Data in an OLTP System

List accounts owned by Tom Franklin

```
SELECT a.cust_id, a.cust_name,
       b.acct_id,
       c.descr as "Account Status",
       d.descr as "Account Type"
  FROM   customers a, accounts b,
         account_states c, account_types d
 WHERE  a.cust_name = 'Tom Franklin' AND
        a.cust_id = b.cust_id AND
        b.acct_state = c.acct_state AND
        b.acct_type = d.acct_type;
```

This query requires a join between two larger tables

cust_id	cust_name	acct_id	Account Status	Account Type
1	Tom Franklin	202	Active	Savings - regular
1	Tom Franklin	201	Active	Checking - regular

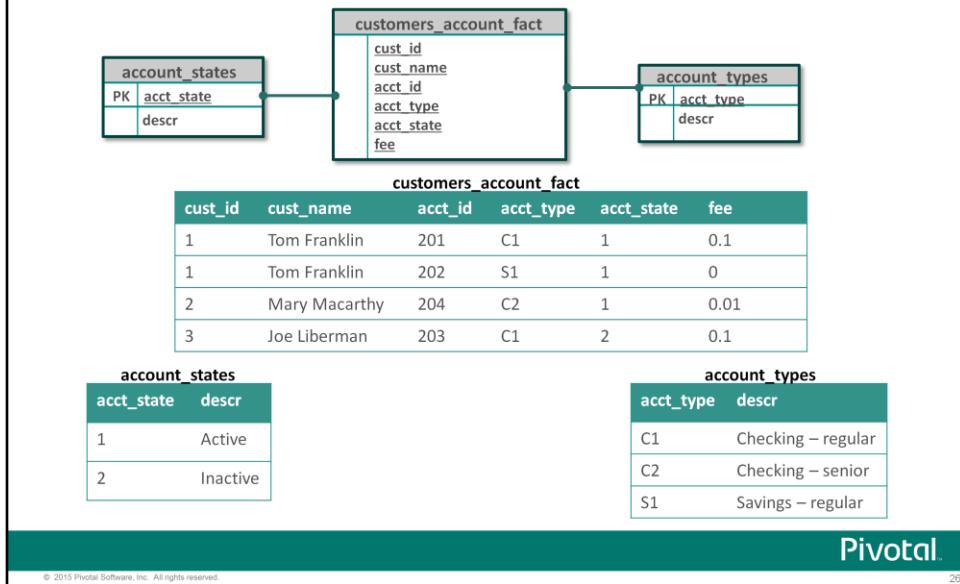
(2 rows)

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

The example highlights the number of joins that are required to extract information on the types of accounts that Tom Franklin owns. It requires that the account and customer tables are joined, along with any other definition tables. The definition tables tend to be smaller, but the accounts and customer tables will be large.

## Data Modeling Example: Denormalized Data used in Data Warehouses



Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

26

When denormalized, data is grouped together to form tables with larger number of columns. The purpose of this is to optimize the read performance of the database. It potentially introduces redundant data, but reduces the needs to perform joins across multiple tables, particularly large tables.

In this example, the customer and account tables have been merged into a single fact table to reduce the number of joins required to retrieve data from the database. The fee column has also been added to the `customers_account_fact` table because it is a measurement that could change over time. The definition tables that describe aspects of the data are now known as dimensional tables. By combining data into a single fact table, there is a greater chance that the data is physically stored together, reducing the I/O that can be introduced with scanning across multiple disks.

## Data Modeling Example: Denormalized Data used in Data Warehouses

List accounts owned by Tom Franklin

```
SELECT a.cust_id, a.cust_name, a.acct_id,
       c.descr as "Account Status",
       d.descr as "Account Type"
  FROM   customers_account_fact a,
         account_states c,
         account_types d
 WHERE  a.cust_name = 'Tom Franklin' AND
        a.acct_state = c.acct_state AND
        a.acct_type = d.acct_type;
```

Large tables have been combined, no longer requiring a join between two large tables

cust_id	cust_name	acct_id	Account Status	Account Type
1	Tom Franklin	202	Active	Savings - regular
1	Tom Franklin	201	Active	Checking - regular

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

27

The act of placing all related facts into a single fact table means there is no longer a need to perform a join on two potentially large tables. In a data warehouse, data may be redundant as it is duplicated in multiple tables. Some customer information may be pulled into another fact table, if necessary, to reduce the I/O that would result from joining multiple large tables to generate reports. By placing the data into a single table, data is more likely to be physically co-located, reducing the number of I/Os necessary to retrieve that data.

While the example shown here may not reflect the type of information an analyst would be looking to obtain from a data warehouse, the purpose was to highlight the denormalization of data and its potential impact on how queries are written and the speed of the results based on the table structures.

## Suggested Reading

For more in-depth information on data warehousing and the relational models, here are some suggested readings:

- *The Data Warehouse Lifecycle Toolkit* by Dr. Ralph Kimball, ISBN 0-471-25547-5
- *Building the Data Warehouse* by William H. Inmon, ISBN 0-471-56960-7
- *The Relational Model* by E.F. Codd, ISBN 0-201-14192-2

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

28

For more information on data warehousing and relational models, reference the following books:

- *The Data Warehouse Lifecycle Toolkit* authored by Dr. Ralph Kimball
- *Building the Data Warehouse*, by William H. Inmon
- *The Relational Model*, by E.F. Codd

While Dr. Ralph Kimball and William H. Inmon represent two separate camps on the discussion of data warehousing, there is no right or wrong in these ideas. They merely represent two different philosophical approaches.

# Module 1: Greenplum Fundamental Concepts

## Lesson 1: Summary

During this lesson the following topics were covered:

- Greenplum Database description and overview
- Big Data and data warehouse
- Differentiating between OLTP and OLAP systems
- Basic elements of a data warehouse
- The role that ETL and ELT plays in data warehousing
- Commonly used methodologies in a data warehouse

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

29

This lesson provided an overview of the Greenplum Database, along with the overall goal and benefits of the product. Before going into great depths into the database, we examined the concepts of Big Data and the data warehouse, differentiating between OLTP and OLAP systems, a combination of which may be found in many organizations and businesses. The basic elements and components of a data warehouse was defined and the role in ETL compared to ELT was also described. Finally, an overview of the data methodologies used in data warehousing was described at a high level.

# Module 1: Greenplum Fundamental Concepts

## Lesson 2: Greenplum Concepts, Features, and Benefits

In this lesson, you examine the main features and benefits offered in Greenplum Database.

Upon completion of this lesson, you should be able to:

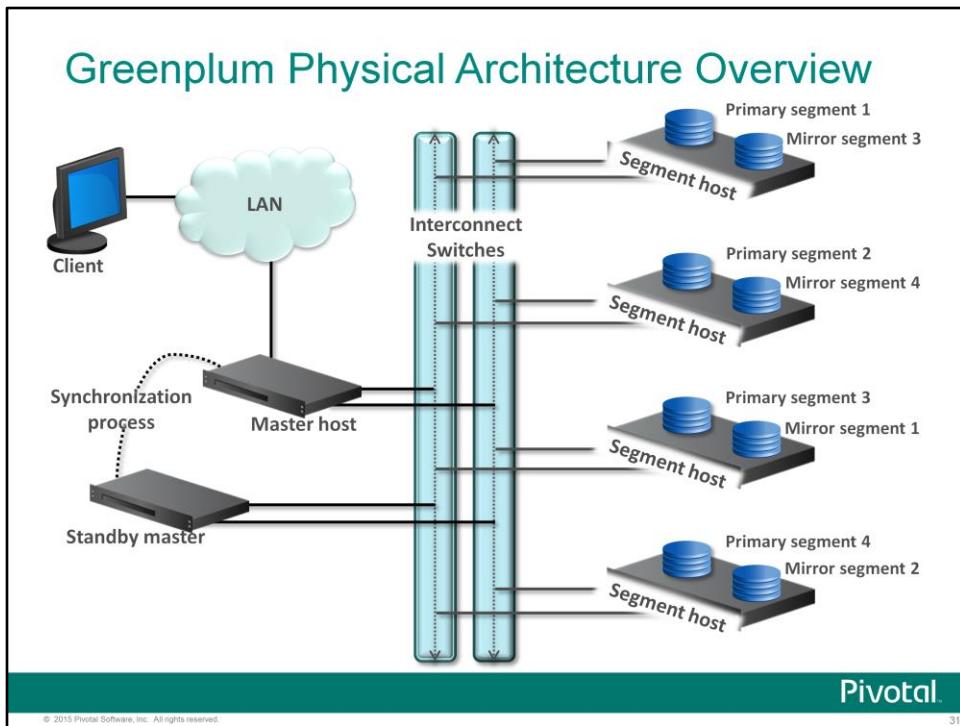
- Examine the physical architecture of Greenplum
- Describe the features of Greenplum
- Identify the major components within the Greenplum architecture
- Identify benefits from implementing a Greenplum solution

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

30

In this lesson, you explore the features and benefits offered in Greenplum Database. You also examine the high-level architecture to understand why Greenplum Database successfully handles mission critical analytics.



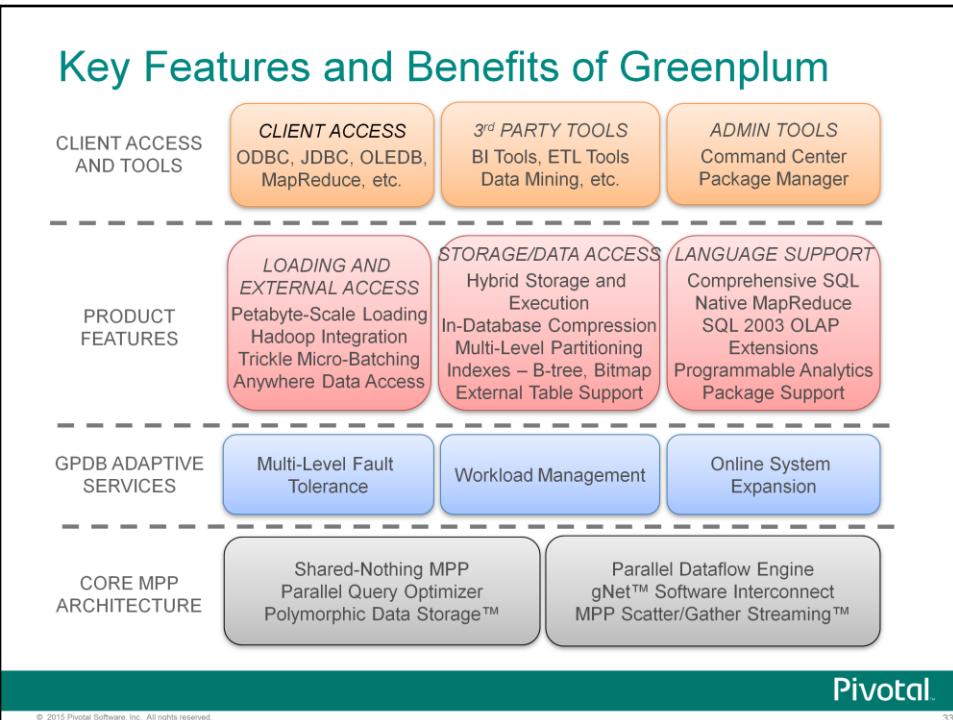
Before examining the features of Greenplum, let us quickly examine the physical architecture of the Greenplum environment and define terminology you will see as you proceed through the course.

A Greenplum environment, or a cluster, consists of the following components:

- **Master host** – The master host is the server responsible for accepting and distributing queries. It is the entry point into the Greenplum environment. While the master itself does not store data, it provides a single abstracted view to all of the data across the segments.
- **Standby master** – The standby master is a warm standby server that is activated when the master host is unavailable.
- **Node or segment host** – The node, or segment host, runs database server processes known as segment instances. It supports one or more segments, depending on the number of processor cores present on the node.

## **Greenplum Physical Architecture (continued)**

- **Primary Segment** – A primary segment stores a distinct portion of the data and is responsible for handling queries. Depending on your hardware configuration, you may have one or more segments per node. Each time data is loaded into the database, a distribution algorithm is used to determine which segments will store what data. When a query is sent to the master, the master develops a query plan and sends this plan to the segments. Each segment is responsible for executing the query on its set of data.
- **Mirror segment** – The mirror segment is a standby segment that is activated should its corresponding primary segment no longer be available.
- **Interconnect switches** – The interconnect switches are the heart of the communication in the Greenplum environment.



Greenplum is a shared-nothing, massively parallel processing (MPP) architecture designed for business intelligence and analytical processing. It is one of the first open-source databases, based on PostgreSQL, that was made available to enterprise environments. Built to support the next generation of Big Data, Greenplum manages, stores, and analyzes terabytes to petabytes of data, with vastly improved performance over traditional relational database management system products.

The logical architecture represents the major features and benefits Greenplum offers. Starting from the bottom of the illustration, you have:

- Core massively parallel processing architecture
- Greenplum adaptive services
- Product features
- Client access and tools

## Core Massively Parallel Processing Architecture

### CORE MPP ARCHITECTURE

Shared-Nothing MPP  
Parallel Query Optimizer  
Polymorphic Data Storage™

Parallel Dataflow Engine  
gNet™ Software Interconnect  
MPP Scatter/Gather Streaming™

Highlights of the core MPP design are:

- Shared-nothing, MPP design emphasizes parallelism, efficiency, and linear scalability
- Parallel query optimizer selects the best plan for the most efficient query execution
- Polymorphic data storage supports tiered data with storage, execution, and compression settings

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

34

The core massively parallel processing architecture highlights several major features:

- **Shared-nothing, MPP design** – The shared-nothing, massively parallel processing architecture utilized by Greenplum incorporates the highest level of parallelism and efficiency to handle complex business intelligence and analytical processing. Greenplum takes advantage of the available hardware in its environment to ensure that data is automatically distributed and query workload is parallelized. Each unit, or segment, within the environment, acts as a self-contained database management system that owns and manages a distinct portion of the overall data. While the data and execution are parallelized, all nodes within a Greenplum environment work together in a highly coordinated fashion.
- **Parallel query optimizer** – When it receives a query, the master server uses a cost-based optimization algorithm to evaluate a vast number of potential plans and selects the one it believes is the most efficient. It does this by taking a global view of execution across the cluster and factors in the cost of moving data between nodes. By taking a global view, you obtain more predictable results than an approach which requires replanning at each node.
- **Polymorphic data storage** – A polymorphic data storage allows customers to choose the storage, execution, and compression settings to support row or column oriented storage and retrieval. This lends support to tiered or temperature aware data, where you may opt to store older data as column oriented with deep archival compression on one set of disks, while more recent data is stored with fast and light compression.

## Core Massively Parallel Processing Architecture (Cont)

### CORE MPP ARCHITECTURE

Shared-Nothing MPP  
Parallel Query Optimizer  
Polymorphic Data Storage™

Parallel Dataflow Engine  
gNet™ Software Interconnect  
MPP Scatter/Gather Streaming™

- Parallel dataflow engine is the heart of Greenplum Database and processes data in parallel, spanning all segments
- gNet software interconnect optimizes the flow of data among all components in the cluster
- MPP scatter/gather streaming uses a scatter approach in data loading to get data from source systems and a gather approach store data on segments

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

35

- **Parallel dataflow engine** – The work of processing and analyzing data is performed in the parallel dataflow engine, the heart of Greenplum Database. This optimized parallel processing infrastructure processes data as it flows from disk, external sources, or other segments over the interconnect. The engine is inherently parallel and spans all segments of a cluster. It can effectively scale to thousands of commodity processing cores. It is highly optimized at executing both SQL and MapReduce in a massively parallel manner. It has the ability to directly execute all necessary SQL building blocks, including performance-critical operations such as hash-join, multistage hash-aggregation, SQL 2003 windowing, and arbitrary MapReduce programs.
- **gNet software interconnect** – One of the most critical components in Greenplum, the gNet interconnect optimizes the flow of data to allow for continuous pipelining of processing without blocking on all nodes of a system. It leverages commodity Gigabit Ethernet and 10GigE switch technology to efficiently pump streams of data between motion nodes during query plan execution. It utilizes pipelining, the ability to begin a task before its predecessor task has completed, to ensure the highest-possible performance.
- **MPP scatter/gather streaming** – Using the MPP scatter/gather streaming, Greenplum is able to achieve data loads of more than 4 terabytes per hour with negligible impact on concurrent database operations. Using a parallel-everywhere approach to data loading, data is scattered from all source systems across hundreds or thousands of parallel streams to all nodes in the cluster. Each node in the cluster simultaneously gathers the data it is responsible for.

## Greenplum Database Adaptive Services

GPDB ADAPTIVE SERVICES

Multi-Level Fault Tolerance

Workload Management

Online System Expansion

To support scalability, changing workloads, and data protection, the following features are inherent in Greenplum:

- Multi-level fault tolerance allows Greenplum to continue operating with hardware and software failures
- Workload management lets an administrator distribute the workload
- Online system expansion lets Greenplum continue operating while hardware is added

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

36

Scalability, workload management, and fault tolerance are features that allow Greenplum to adapt to a changing environment, increase uptime, and scale storage and compute power.

- **Multi-level fault tolerance** – Using multiple levels of fault tolerance and redundancy, Greenplum can continue operating in the face of hardware and software failures. Mirrors for the master and segments help to protect against data loss as well as database operation loss. The interconnect provides redundant access to all nodes, the master, standby master, and any other components connected to the switch.
- **Workload management** – The database administrator has administrative control over determining system resources to users and queries. User-based resource queues automatically manage the flow of work to the databases from defined users. Query prioritization allows control of runtime query prioritization to ensure queries have appropriate access to resources. This allows you to prevent one query from hogging all system resources and potentially starving other queries out of these resources. This also allows you to redistribute resources based on user loads.
- **Online system expansion** – Servers can be added to not only increase storage capacity, but also to increase processing power and loading performance. The database can remain online while the expansion process takes place in the background. Due to the implementation of the shared nothing, MPP design, increasing the number of nodes in the cluster increases performance and capacity linearly for Greenplum. Support for dynamic provisioning means you can add onto existing configurations without having to replace existing configurations.

## Product Features – Loading and External Access

### PRODUCT FEATURES

#### LOADING AND EXTERNAL ACCESS

- Petabyte-Scale Loading
- Hadoop Integration
- Trickle Micro-Batching
- Anywhere Data Access

#### STORAGE/DATA ACCESS

- Hybrid Storage and Execution
- In-Database Compression
- Multi-Level Partitioning
- Indexes – B-tree, Bitmap
- External Table Support

#### LANGUAGE SUPPORT

- Comprehensive SQL
- Native MapReduce
- SQL 2003 OLAP Extensions
- Programmable Analytics Package Support

Access to data is achieved with the following features:

- Petabyte-scale loading uses the MPP Scatter/Gather to load and unload data
- Hadoop integration provides co-processing of structured and unstructured data
- Trickle micro-batching supports loading in a continuous stream so that data can be loaded more frequently
- Anywhere data access lets you access and make available data external to the Greenplum database

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

37

To load data and access external data, Greenplum offers the following features:

- **Petabyte-scale loading** – Using the MPP Scatter/Gather streaming technology, Greenplum can perform high-performance loading and unloading of data. With each additional node in the cluster, the speed at which the loads, parallel data digest, and unloads, parallel data output, are performed increases linearly.
- **Hadoop integration** – Hadoop provides a solid method of processing unstructured data. Greenplum Database provides high performance parallel import and export of data from Hadoop clusters and filesystems.
- **Trickle micro-batching** – When loading a continuous stream, trickle micro-batching allows data to be loaded at frequent intervals, such as every five minutes, while maintaining extremely high data ingest rates. This is extremely useful to companies such as the NYSE or other trade and securities institutions.
- **Anywhere data access** – Data external to the Greenplum database can be accessed, regardless of their location, format, or storage medium. Greenplum allows you to define external tables that access this data and makes it available for reads or writes.

## Product Features – Storage and Data Access

<b>PRODUCT FEATURES</b>	<b>LOADING AND EXTERNAL ACCESS</b> Petabyte-Scale Loading Hadoop Integration Trickle Micro-Batching Anywhere Data Access	<b>STORAGE/DATA ACCESS</b> Hybrid Storage and Execution In-Database Compression Multi-Level Partitioning Indexes – B-tree, Bitmap External Table Support	<b>LANGUAGE SUPPORT</b> Comprehensive SQL Native MapReduce SQL 2003 OLAP Extensions Programmable Analytics Package Support
-------------------------	--	---	--

Data storage and access features include:

- Hybrid storage and execution lets a DBA select storage, execution, and compression settings for data
- In-database compression provides increased performance and reduced storage
- Multi-level partitioning provides flexible partitioning of tables
- Index support is provided for B-tree, bitmap, and GiST indexes
- External tables provide data loading and unloading to external points

© 2015 Pivotal Software, Inc. All rights reserved.
38

Pivotal

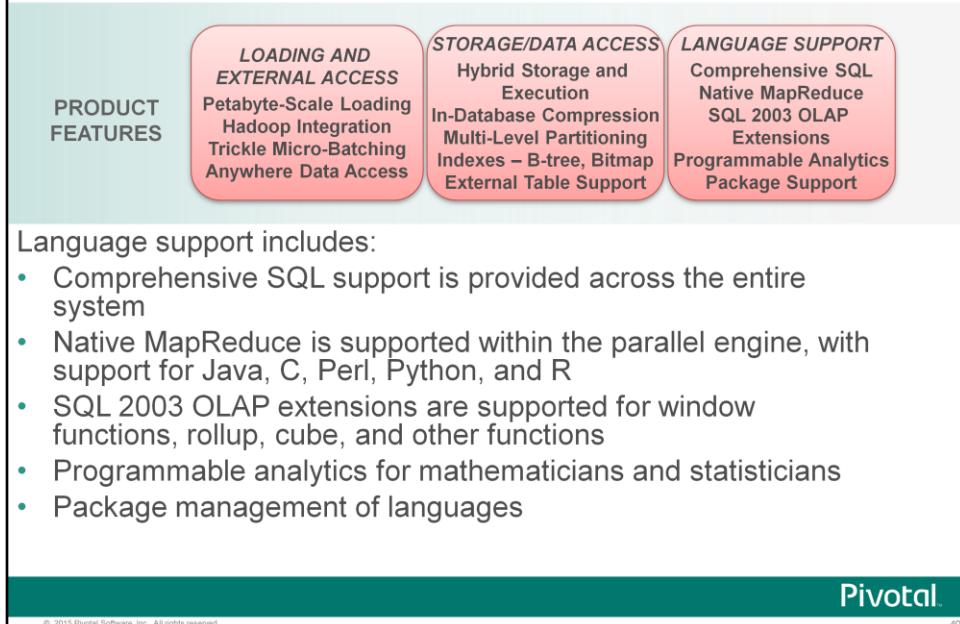
Data storage and access features include:

- **Hybrid storage and execution** – For each table or partition of a table, the database administrator can select the storage, execution, and compression settings that suit the way that table will be accessed. This feature includes the choice of row- or column-oriented storage and processing for any table or partition. This leverages Greenplum's Polymorphic Data Storage technology and allows for tiered storage, where the database administrator can define which data will have lighter compression to allow for faster access and which is not accessed as frequently.
- **In-database compression** – Increased performance and reduced storage can be achieved with in-database compression. By reducing the amount of disk space data takes up, you see an increase in effective I/O performance. In-database compression allows for the storage of years of data, economically. This allows you to get into a true discussion of compliance, e-discovery, and regulatory issues, where you can pull data from previous years quickly. You may not be able to query as quickly, depending on your storage plan, but you will be able to more quickly access data that hasn't been moved off to slow storage or tape.
- **Multi-level partitioning** – With multi-level partitioning, you have flexible partitioning of tables based on date, range, or value. Partitioning is specified using DDL and allows an arbitrary number of levels. The query optimizer will automatically prune unneeded partitions from the query plan.

## **Product Features – Storage and Data Access (continued)**

- **Index support** – Greenplum provides support for a range of index types, including B-tree, bitmap, and Generalized Search Tree (GiST) indexes.
- **External Tables** – External tables lets administrators load data into the Greenplum Database using externally available files, web sites, and pipes. Data is unloaded in the same way to external resources. This extends the methods available for loading and unloading data to and from Greenplum Database.

## Product Features – Language Support



Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

40

Powerful language support gives developers flexibility in how to approach Greenplum. Language support is provided for:

- **Comprehensive SQL** – Greenplum offers comprehensive SQL-92 and SQL-99 support with SQL 2003 OLAP extensions. All queries are parallelized and executed across the entire system.
- **Native MapReduce** – MapReduce has been proven as a technique for high-scale data analysis by Internet leaders such as Google and Yahoo. Greenplum Database natively runs MapReduce programs within its parallel engine. Languages supported include Java, C, Perl, Python, and R.
- **SQL 2003 OLAP Extensions** – Greenplum provides a fully parallelized implementation of SQL recently added OLAP extensions. This includes full standard support for window functions, rollup, cube, and a wide range of other expressive functionality.
- **Programmable Analytics** – With programmable analytics, Greenplum offers a new level of parallel analysis capabilities for mathematicians and statisticians, with support for R, linear algebra, and machine learning primitives. Greenplum also provides extensibility for functions written in Java, C, Perl, or Python.
- **Package Support** – Greenplum also incorporates package support to provide turn key analytic extensions that allows you to more easily manage your language extensions.

## Client Access and Tools

CLIENT  
ACCESS  
AND TOOLS

**CLIENT ACCESS**  
ODBC, JDBC, OLEDB,  
MapReduce, etc.

**3<sup>rd</sup> PARTY TOOLS**  
BI Tools, ETL Tools  
Data Mining, etc.

**ADMIN TOOLS**  
Command Center  
Package Manager

Client access and tools are provided by:

- Client access tools and drivers
- Third party tools used for BI, ETL, data mining, and data visualization
- Administrative tools include Greenplum Command Center, Greenplum Package Manager

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

41

Users and administrators have access to Greenplum through a variety of tools, including:

- **Client access** – Tools and drivers such as ODBC, JDBC, and OLEDB, can be used to access the Greenplum database. MapReduce lets you process large data sets with a parallel distributed algorithm.
- **Third party tools** – Business intelligence tools, ETL tools, applications for data mining and data visualization can also gain access to the Greenplum database. Health monitoring and alert notifications are provided with email and simple network management protocol (SNMP) notifications.
- **Administrative tools** – Greenplum Command Center lets administrators manage and monitor the state of the system and workloads, including system metrics and query details on the system. Command Center provides a dashboard for managing and monitoring the system and database, along with queries. You can drill down into a query's detail and plan to understand its performance.  
Greenplum Package Manager lets you install additional supported languages through a package management utility.  
You can access your environment with tools such as pgAdmin 3. pgAdmin 3 is the most popular and feature-rich Open Source administration and development platform for PostgreSQL.

## Benefits of Greenplum

The infographic consists of six benefit cards arranged in two rows of three. Each card features an icon and a brief description.

- Faster performance**: Represented by a yellow lightning bolt icon.
- Real-time analytics**: Represented by a clock with a green circular arrow icon.
- Flexibility and control**: Represented by a grid of nine colored squares (green, orange, blue, red, yellow) each containing a white 'i' symbol.
- Centralized management**: Represented by a globe with a central server icon.
- Enterprise class reliability**: Represented by a globe with a red double-headed arrow icon.
- Linear scalability**: Represented by a series of three red spheres connected by a dashed line.

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

Customers who implement Greenplum gain benefits in:

- **Faster performance** – Increased resources support increased workload. You can see performance gains of 10 to 100 times faster than traditional data warehouse technologies.
- **Real-time analytics** – Greenplum enables sophisticated queries and ad-hoc analysis with multiple terabytes to petabytes of data. With OLAP queries, you can perform advanced queries without having to use third party tools.
- **Flexibility and control** – You can decide whether to choose EMC's hardware or your own. This gives you control over the choice of hardware and operating systems, as well as the ability to add capacity and therefore performance, inexpensively.
- **Centralized management** – Centralized management allows ease of configuration through a single central location – the master. In the end, centralized cluster management and administration lowers total cost of ownership (TCO).
- **Enterprise class reliability** – High availability, mirroring on segments and standby and hardware-level mirroring. With multiple levels of redundancy and fail-over, this minimizes downtime.
- **Linear scalability** – Nodes can be expanded on an as needed basis. This allows for predictable, linear performance gains and capacity growth. It is recommended that Professional Services is involved when expanding nodes. Expanding nodes involves reconfiguring the database to make immediate use of the hardware with preexisting data and newly stored data.

# Module 1: Greenplum Fundamental Concepts

## Lesson 2: Summary

During this lesson the following topics were covered:

- Physical architecture of Greenplum
- Features of Greenplum
- Major components within the Greenplum architecture
- Benefits from implementing a Greenplum solution

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

43

This lesson provided a more in-depth description of the Greenplum Database, including the physical architecture that makes up the database, the major features of the Database, the components within the architecture and how they interact with each other and clients, and the benefits from implementation a Greenplum Database solution within your environment.

# Module 1: Greenplum Fundamental Concepts

## Lesson 3: Greenplum Architecture – Shared Nothing and MPP

In this lesson, you take an in-depth look at the shared-nothing, MPP architecture implemented in Greenplum.

Upon completion of this lesson, you should be able to:

- Describe the shared-nothing, massively parallel processing architecture
- Identify key times when parallelism is implemented in data management, system administration, and monitoring
- List hardware solutions available for Greenplum

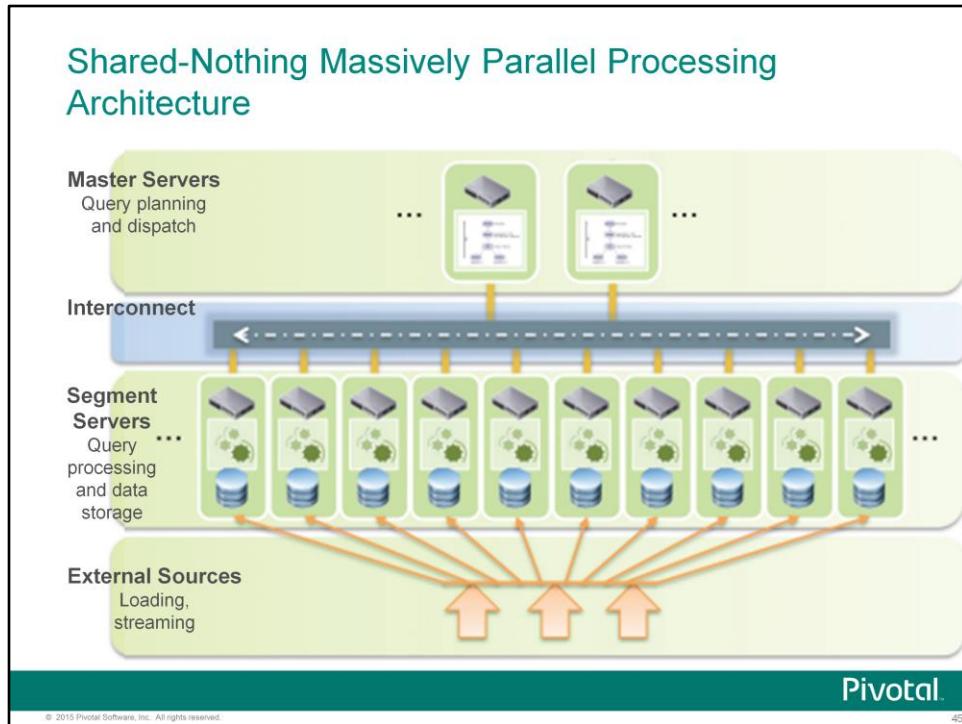
Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

44

In this lesson, you examine the shared-nothing, MPP architecture in greater detail, analyzing how and where parallelism is implemented in Greenplum.

You will also examine the type of hardware solutions that are available for the Greenplum Database.



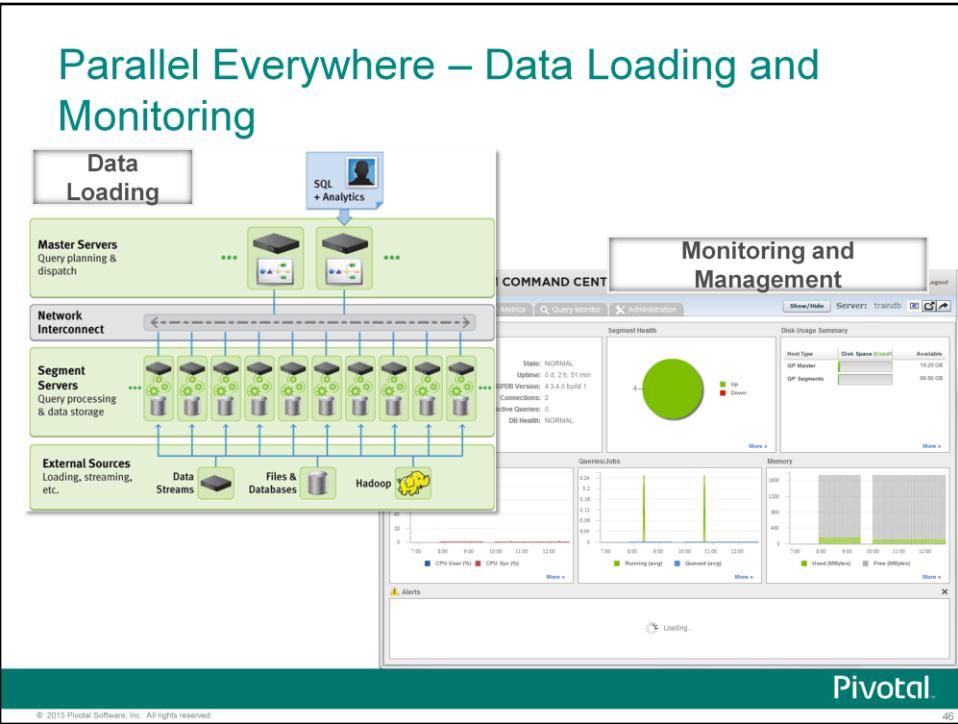
The shared-nothing, massively parallel processing architecture refers to systems with two or more processing units, or segment hosts, which cooperate to carry out an operation. Each segment host has its own:

- Processor
- Memory
- Disk
- Operating system

Each segment host runs its own Greenplum database instance. Greenplum leverages this high performance system architecture to distribute the load of multi-terabyte data warehouses. It uses all of a system's resources in parallel to process a query. In a sense, Greenplum Database is a cohesive database management system comprised of several PostgreSQL database instances acting together.

While Greenplum is based on PostgreSQL 8.2.9, maintaining SQL support, main PostgreSQL features, configuration options, and the same end-user functionality, it extends PostgreSQL by including features for business intelligence workloads, including:

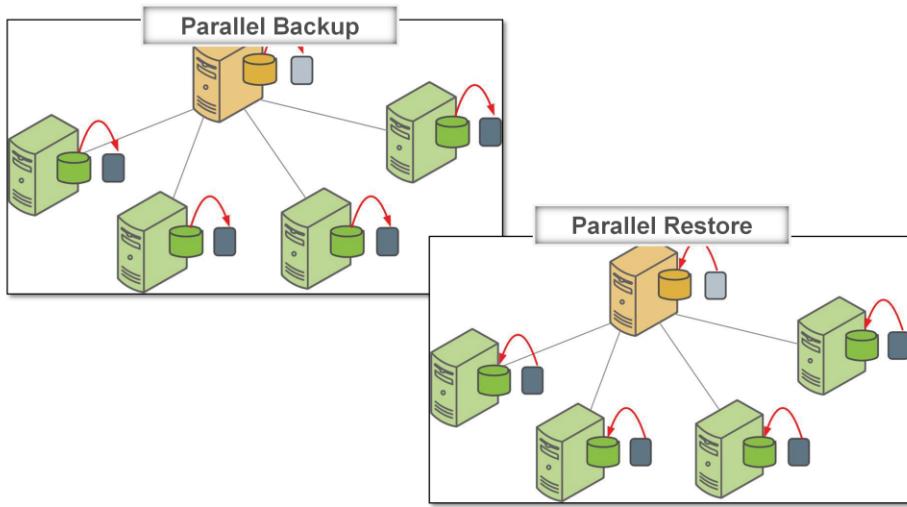
- Parallel data loading
- Implementation of external tables
- Resource management
- Query optimizations
- Storage enhancements



In the shared-nothing, MPP architecture, parallelism is key to the following:

- **Data loading, storage, and querying** – Data loading into the Greenplum databases can come directly through an ETL host connected to the interconnect. The `gpfldist` command, provided with Greenplum, is a custom command that runs an internal HTTP light server and allows Greenplum to connect to the ETL. Data is loaded to all segments simultaneously, using the Scatter/Gather method. The query execution plan is broadcast to all segments, even if they do not contain data. The segments then respond to the query plan with appropriate data. All of the work is performed in parallel: segments work to load, read, and modify the user data in parallel. While some segments may not contain data, the query plan is still issued to all segments.
- **Monitoring and Management** – When monitoring the Greenplum environment, Greenplum Command Center checks the status of the system as a whole. You can obtain the status of each individual Greenplum instance in the cluster, while managing the database and checking on the performance of queries executing in the database. Command Center also lets you manage specific aspects of the database. This includes but is not limited to controlling queue priorities, resource queue management, and database status and availability.

## Parallel Everywhere – Backups and Restores



Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

47

Backup and restore is another function performed in parallel. When you issue the command to backup data, data is compressed and archived onto the appropriate host in parallel. Restores to segments are also retrieved in parallel.

During backup, data is backed up to a local backup directory. The backup is archived in parallel. Restore functions in the same way, retrieving in parallel.

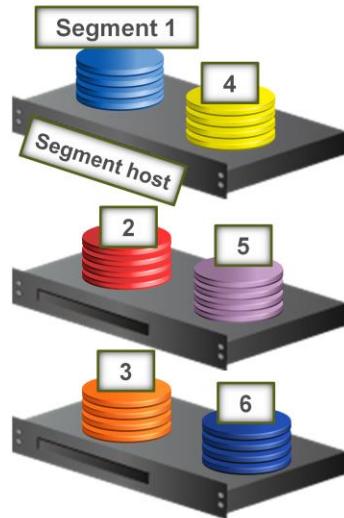
A Data Domain Device is an optimal solution that can be used for backup and restores. Data is backed up, compressed, and moved onto an archive host. Restores are retrieved from the archive host and then uncompressed.

The key to performing a backup is to have enough local storage on the segment host and master host to perform the backups. While Greenplum itself is not necessarily focused on backups, it works well with other software, such as Data Domain, to provide data backups, archival, and restoration services.

All backups and restorations are performed on the master.

## Data Distribution – the Key to Parallelism

OrderNum	Order_Date	Customer_ID
43	Oct 20 2010	12
64	Oct 20 2010	111
45	Oct 20 2010	12
46	Oct 20 2010	64
77	Oct 20 2010	32
48	Oct 20 2010	12
50	Oct 20 2010	32
56	Oct 20 2010	213
63	Oct 20 2010	15
49	Oct 20 2010	111



Pivotal.

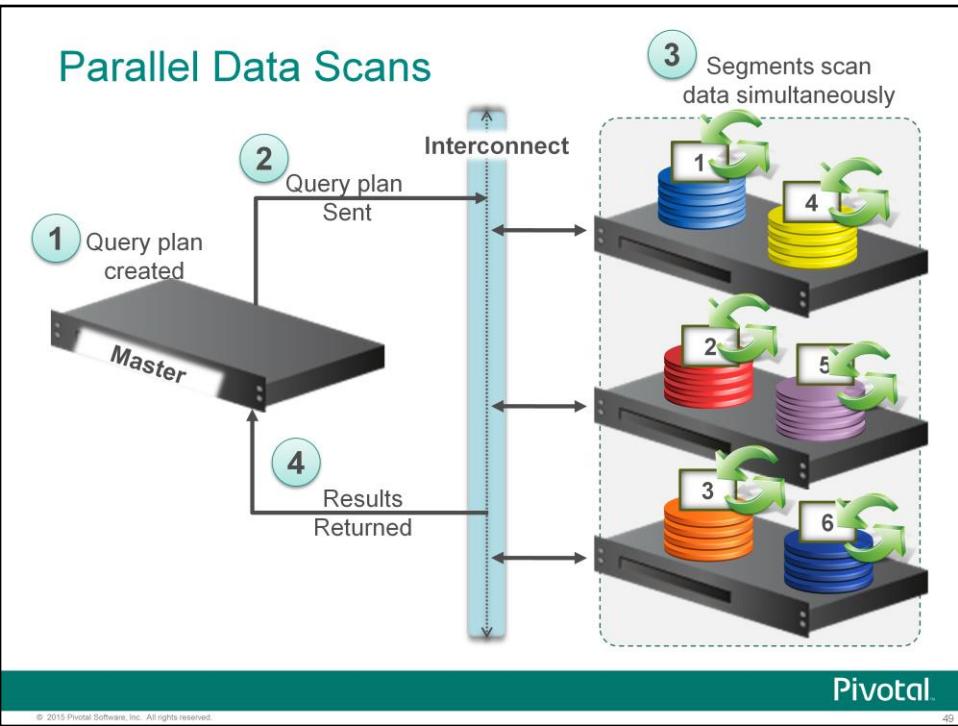
© 2015 Pivotal Software, Inc. All rights reserved.

48

When distributing data, the best scenario is to have the data co-located as much as is possible. The data should be spread as evenly as possible across as many segments as possible. There are several key points to remember when distributing the data:

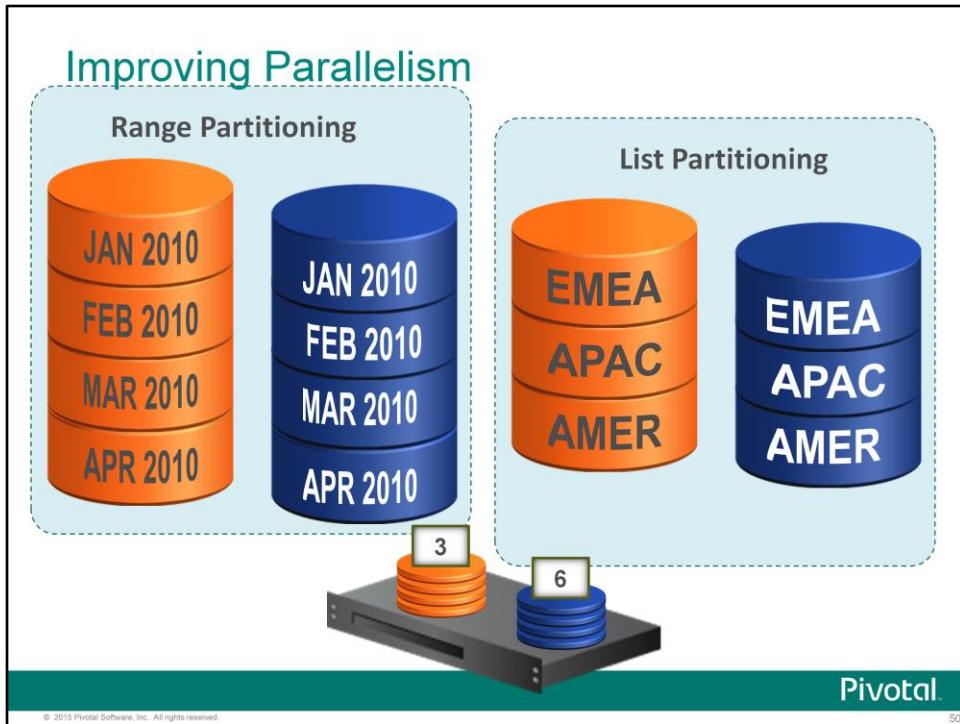
- The distribution key is used to determine how the data is spread across the segments. Choose the distribution key wisely.
- The data type is important for hashing out the distribution policy used for distributing the data. An integer and a BigInt do not hash in the same manner.
- Analyze the data and examine commonly run queries. This helps to determine which column should be used as the distribution key.

In the example shown on the slide, data is distributed on the distribution key, Customer\_ID. Choosing Customer\_ID as the distribution key more evenly spread the data across all segments. If you had chosen the Order\_Date column, all of the data, based on this sample set, would be written to a single segment.



When a query is sent to the master server:

1. The master server develops a query plan by taking a global view of execution across the entire cluster and factors in any movement that must occur, such as when joining tables together. This helps to create a more predictable result.
2. The query plan is sent to all of the segments within the cluster.
3. Each segment scans its set of data independently of other segments. This action occurs in parallel.
4. Once all segments have completed executing the query based on the query plan, the results are returned to the master, which in turn returns the results to the client.



To improve the results achieved with parallelism, Greenplum supports data partitioning.

Partitioning large tables can reduce the number of rows that needs to be scanned when performing a data scan for a query. Data is broken into bucket on each segment, based on a filter. When a data scan is performed, only the buckets that match the query are searched. This reduces the scan size and therefore the work that each segment must perform to respond to a query.

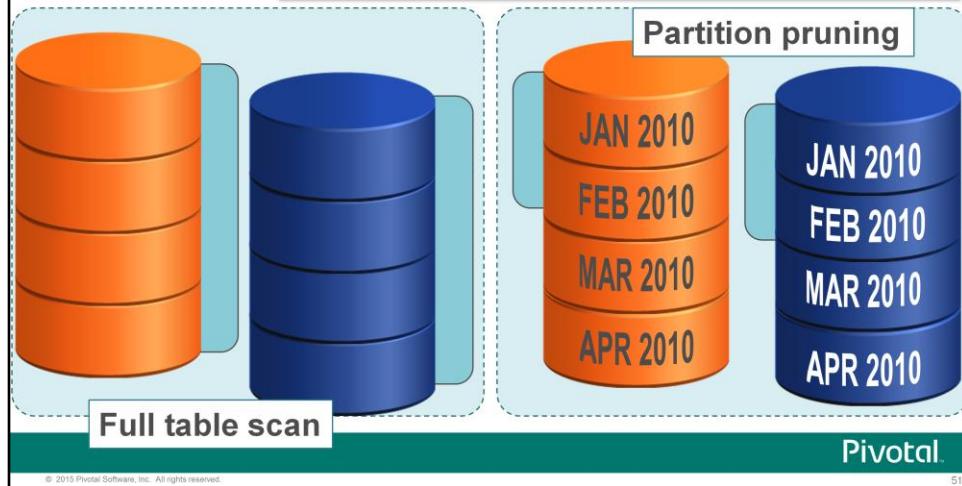
Two types of partitioning are supported:

- **Range partitioning** – Data is partitioned based on a numerical range, such as date or price.
- **List partitioning** – Data is partitioned based on a list of values, such as sales territory or a product line.

**Note:** Partitioning is above and beyond the hash-based distribution and allows the system to scan just the subset of buckets that might be relevant to the query

## Comparing a Full Table Scan to Partition Pruning

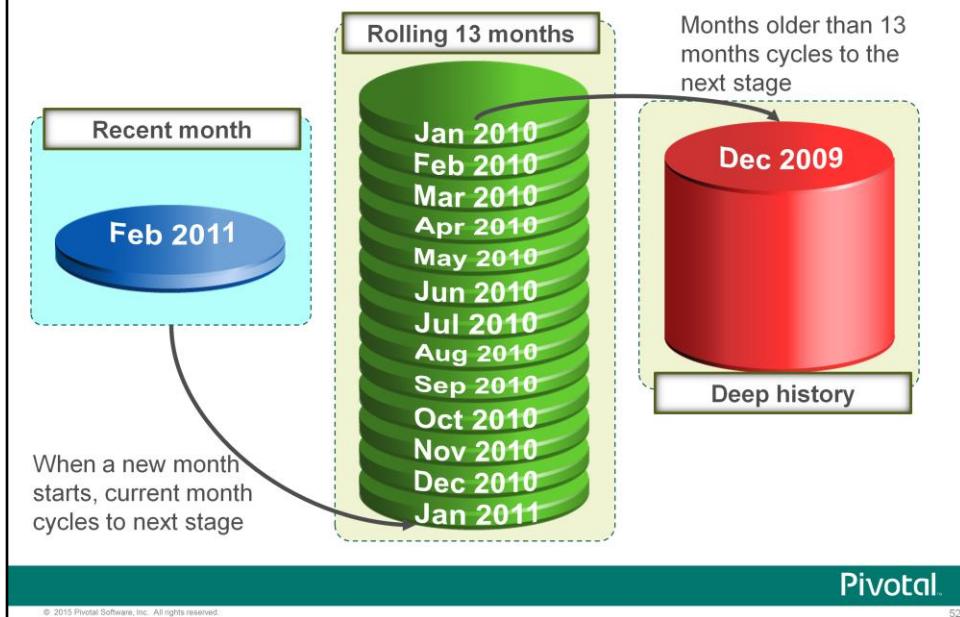
```
SELECT COUNT(*)  
  FROM orders  
 WHERE order_date >= 'Jan 20 2010'  
   AND order_date < 'Feb 27 2010'
```



When presented with a query, each segment scans its data to yield the appropriate results. If the table is not partitioned, a full table scan is performed.

If the table is partitioned, only the buckets that match the criteria are scanned. This can greatly reduce the amount of work that must be performed by each segment.

## Historical Data Management



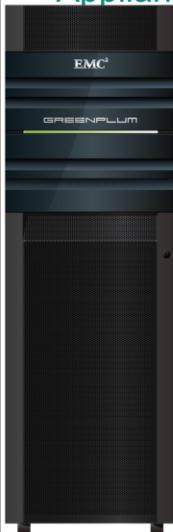
Partitions do not have to be the same size or orientation, which can be column or row-oriented. You can partition your data based on a selected time frame.

This example shows a rolling management scheme, where 13 months of data are maintained at a time. The scheme is as follows:

- Anything older than 13 months is moved to deep history.
- The current month is maintained *in memory* where access is fastest.
- All in between months are maintained at a second level of storage

The strategy is customizable based on your organization's needs.

## Hardware Solutions – Pivotal Data Computing Appliance



**Extreme Performance**  
Optimized for fast query execution and unmatched data loading

**Rapidly Deployable**  
Purpose-build data warehousing appliance

**Modular Design**  
Offers modular solution for structured data, unstructured data, and ETL

**Highly Available**  
Self healing and fully redundant

**Elastic Scalability**  
Expand capacity and performance online

**Reduced TCO**  
Consolidate Data Marts for lower costs

**Private Cloud Ready**  
Data and computing are automatically optimized and distributed

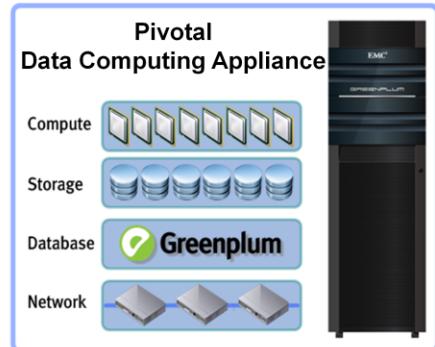
**Advanced Backup and DR**  
Leverage industry-leading Data Domain backup and recovery with SAN mirroring solutions

Pivotal

The Pivotal Data Computing Appliance (DCA) is a purpose-built, highly scalable parallel data warehousing appliance that architecturally integrates database, compute, storage and network into an enterprise class easy-to-implement system. It offers a modular solution for structured data, unstructured data, and Greenplum partner applications for Business Intelligence and ETL services. The Pivotal DCA:

- Brings the processing power of an MPP architecture
- Delivers the fastest data loading capacity, and the best price/performance ratio in the industry
- Does not have the complexity and constraints of proprietary hardware
- Provides a flexible infrastructure to support expansion based on growing business needs

## Benefits of an Appliance Approach



Benefits of implementing a DCA solution:

- Easy implementation
- Price / Performance leadership
- Private Cloud Ready
- Massively Parallel, Scalable
- Next generation analytic processing
- Enterprise-proven feature set

Pivotal

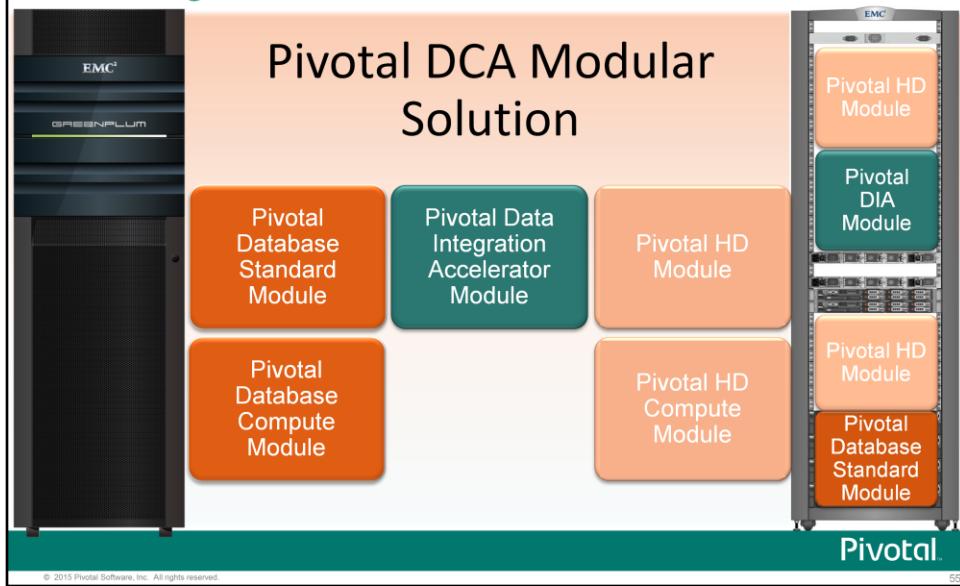
© 2015 Pivotal Software, Inc. All rights reserved.

54

While you can implement any reference architecture within your Greenplum environment, the benefits of using an appliance has great appeal for scalability, price point, and support. Benefits of implementing the Pivotal DCA include:

- A seamlessly installed and integrated system that provides lower total cost of ownership (TCO) and higher return on investment (ROI).
- Multiple DCA solutions that provide price and performance leadership for data warehouse hardware solutions
- The industry's first enterprise data cloud platform to support virtualized data warehouses and an analytic infrastructure
- The MPP approach and flexible scalability to meet increasing service level demands make it a price/performance leader in today's market. It supports the Greenplum Database shared-nothing, MPP architecture by moving the processing dramatically closer to the data and users, allowing resources to process every query in a fully parallel manner. All storage connections are used simultaneously while data flow is efficiently managed between resources when performing moves as a result of a query plan.
- Next generation analytic processing lets you quickly analyze Big Data to address the fast-paced environment
- High availability, fault tolerance and the robust analytic processing of large streams of data further support the overall benefits of a data computing appliance approach to data warehousing and business intelligence.

## Expanding Modular Support – The Total Package



Through its new modular support, the Pivotal DCA supports at a minimum, one Greenplum Database Module, and three optional Greenplum Database modules, to allow you to customize your environment as you need. The Pivotal HD option supports a minimum of a two Pivotal HD module – the first module contains the master servers while the second module provides worker servers.

You can get results faster by using an integrated appliance that offers optimized performance, ease of deployment, increased system monitoring and manageability, and a reduced footprint. The Pivotal DCA modules greatly simplify the expansion of capacity and performance of the Greenplum Database (analytic database) and Pivotal HD (Apache Hadoop) portions of the systems. This data management appliance delivers maximum flexibility and scalability for organizations that are tackling terabyte- to petabyte-scale data opportunities.

All modules utilize the Pivotal DCA interconnect to provide high-speed, high-performance, low-latency connections. This speed is maintained whether a single rack or multiple racks are employed in the cluster.

Pivotal DCA Hardware Configurations	
	Pivotal DB Compute Module 4 x 2U servers with 24 x 300-GB disk drives per server 9 TB uncompressed : 36 TB user data (compressed) 256 GB of Memory : 64 CPU cores
	Pivotal DB Standard Module 4 x 2U servers with 24 x 900-GB disk drives per server 27.5 TB uncompressed : 110 TB user data (compressed) 256 GB of Memory : 64 CPU cores
	Pivotal DIA Module 2 x 1U servers with 6 x 300-GB disk drives 1.8 TB Data 256 GB of Memory : 64 CPU cores
	Pivotal HD Module 4 x 2U servers with 12 x 3-TB disk drives per server 36 TB uncompressed : 144 TB user data 256 GB of Memory : 64 CPU cores
	Pivotal HD Compute Module 2 x 1U servers 128 GB of Memory : 32 CPU cores
Pivotal	

The Pivotal DCA is a modular solution that lets you incorporate Greenplum Database, Pivotal HD, and third-party solutions in an expandable rack solution. Configurations support ranges from a single Pivotal Database, formerly Greenplum Database, or Pivotal HD module to a 12-rack solution with a mix of modules.

Supported modules include:

- Pivotal Database Compute Module offers is a cost-effective solution for those who require large computational power but less storage needs.
- Pivotal Database Standard Module, which supports the Greenplum Database, and offers the best price to performance ratio with linear scalability. Additional modules can expand the system to accommodate the growing needs of the organization. A full rack offers a scan rate of 40 GB/sec with a data load rate of 16 TB/hour.
- Pivotal Data Integration Accelerator module provides fast integration for partner analytics applications to the Pivotal Data Computing Appliance. With its direct access to the Pivotal DCA 10 gigabit per second interconnect, applications such as Informatica, Talend, and Pentaho can take advantage of the MPP architecture to integrate data into the Greenplum environment.

## **Pivotal DCA Hardware Configurations (continued)**

- Pivotal HD and Pivotal HD Compute modules support unstructured data with Hadoop. The Pivotal HD module supports in-rack storage and computation for Hadoop services, while the Pivotal HD Compute module provides users with the opportunity to extend and scale storage needs with Isilon OneFS scale-out NAS for remote storage.

Each module has CPUs, memory, disk I/O, and interconnects optimally balanced for Greenplum Database, Pivotal HD, and party software for Business Intelligence, ETL, analytics, and data visualization. With the exception of the Pivotal HD Compute module and the DIA which both have two servers, all other modules support four servers each. A cluster can span 1 to 48 modules, yielding 36TB to 5PB of total user data capacity. Total compute capacity ranges from 64 to over 3000+ CPU cores with a multi-rack solution. Various configurations may limit the number and types of modules you incorporate into the final solution.

## Expanding Modular Support – Pivotal Data Integration Accelerator



### Rapid Deployment

Integrates Greenplum data loading into a single, easy-to-implement solution

### Predictable Performance

Packaged, pre-tuned, and simplified for data loading

### Tight Integration

Integrates into an existing Pivotal DCA solution

### Engineered for Data Loading

Manages data flow using MPP Scatter/Gather Streaming technology

### Enterprise High Availability

RAID protection at the disk level

### Centralized Monitoring

Managed and monitored with the Greenplum Command Center

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

58

The Pivotal Data Integration Accelerator, otherwise known informally as the ETL module, is a purpose-built, open systems data accelerator that architecturally integrates data loading software, server, storage, and networking into a single, easy-to-implement module. This module is installed directly into the Pivotal Data Computing Appliance rack and provides predictable performance through packaging and pre-tuning for data loading activities.

The module allows implementers to install partner analytics applications and data management applications into the Pivotal DCA. Pivotal applications, such as Greenplum Command Center, Pivotal Command Center, Pivotal Chorus, and Pivotal VRP can be installed in this module, reducing the potential impact on database modules in the rack or the need to use separate servers to support the applications.

Its main features include:

- Tight integration within a Pivotal DCA to provide an end-to-end solution
- Parallel data loading solution by using the MPP Scatter/Gather Streaming technology to send data from all nodes on the DIA to every segment server of the database
- Disk RAID protection to provide mission-critical support to enterprises
- Configuration support through the Greenplum setup and configuration tool
- Management support through Greenplum Command Center

## Expanding Modular Support – Pivotal HD



### Rapid Deployment

Integrates support for processing unstructured data with Hadoop Enterprise

### Tight Integration

Integrates into an existing Pivotal DCA solution

### Engineered for Balanced Loads

Utilizes a parallel flow for reading and writing data

### Enterprise High Availability

RAID protection at the disk level  
Intelligent snapshots  
Mirroring (software replication)

### Centralized Monitoring

Managed and monitored with the Pivotal Command Center

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

59

The Pivotal HD modules fuse Hadoop with the Greenplum Database, offering support for co-processing of both structured data with the Greenplum Database, and unstructured data with enterprise-ready Hadoop infrastructure. The module continues to provide enterprise high-availability with RAID protection, intelligent snapshots and mirroring at the software level for the Greenplum Hadoop Enterprise software.

Centralized monitoring offered with Pivotal Command Center. Pivotal Command Center differs from Greenplum Command Center in that it is focused on monitoring and management Pivotal HD modules and Hadoop services.

## Lab: Preparation and Initialization

In this lab, you familiarize yourself with the lab environment you will be using throughout the course.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

60

In this lab, you will access the lab environment and familiarize yourself with the tools you will use throughout the remainder of the course.

# Module 1: Greenplum Fundamental Concepts

## Lesson 3: Summary

During this lesson the following topics were covered:

- Shared-nothing, massively parallel processing architecture
- Key times when parallelism is implemented in data management, system administration, and monitoring
- Hardware solutions available for Greenplum

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

61

This lesson covered the shared-nothing, massively parallel processing architecture that drives the improved performance and communication found within the Greenplum Database architecture. The lesson also covered the key moments when parallelism is implemented during data management, system administration, and system monitoring of the Greenplum environment, such as during data loads and data scans. Hardware solutions for the Greenplum Database were also presented, specifically the Pivotal Data Computing Appliance and available modules for the appliance.

# Module 1: Greenplum Fundamental Concepts

## Lesson 4: Greenplum Product Overview

In this lesson, you examine the theory of operations for the Greenplum architecture.

Upon completion of this lesson, you should be able to:

- Describe primary architecture and components
- Describe redundant components
- Define key concepts: distributed data and queries

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

62

In this lesson, you:

- Examine the theory of operations for the Greenplum architecture. To do that, you will gain an deeper insight of the components of the architecture introduced earlier and examine how data moves from the client to the segments and back to the client again.
- Identify the components within the architecture that are key to maintaining high availability for the Greenplum environment.
- Define key concepts, including distributed data and queries.

## What Is the Greenplum Database?

The Greenplum Database:

- Is a Massively Parallel Processing (MPP) DBMS
- Is based on PostgreSQL 8.2.15
  - Similar client functionality
  - Additional technology to support parallelism
- Supports additional features for DW and BI
  - External tables / parallel loading
  - Resource management
  - Query optimizer enhancements

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

63

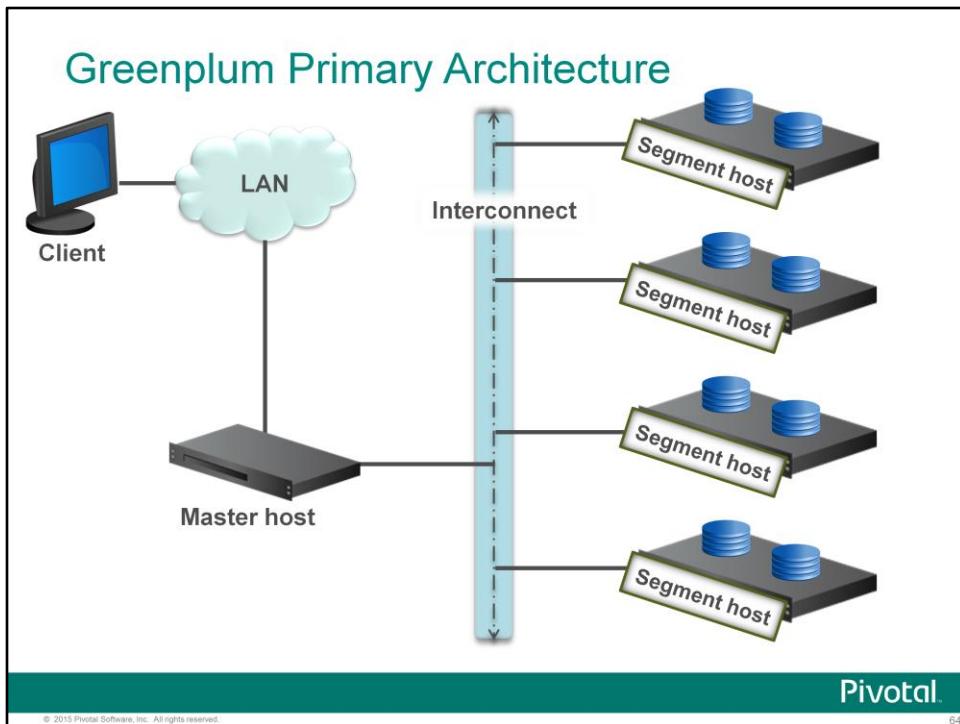
MPP, also known as a *shared-nothing* architecture, refers to systems with two or more units or nodes that cooperate to carry out an operation. Each node has its own memory, operating system, and disks. Greenplum leverages this high-performance system architecture to distribute the load of large volume of data warehouses, and is able to use all of a system's resources in parallel to process a query.

Greenplum Database is:

- Several PostgreSQL database instances acting as one cohesive database management system.
- Based on Postgres 8.2.15 and, in most cases, is very similar to current-day PostgreSQL with regards to SQL support, features, configuration options, and end-user functionality. Database users interact with Greenplum DB as they would a regular PostgreSQL DBMS.

The internals of PostgreSQL have been modified or supplemented to support the parallel structure of Greenplum DB. For example the system catalog, query planner and optimizer, query executor, and transaction manager components have been modified and enhanced to be able to execute queries in parallel across all of the database instances at once. The interconnect component enables communication between the distinct PostgreSQL instances and makes the system behave as one logical database.

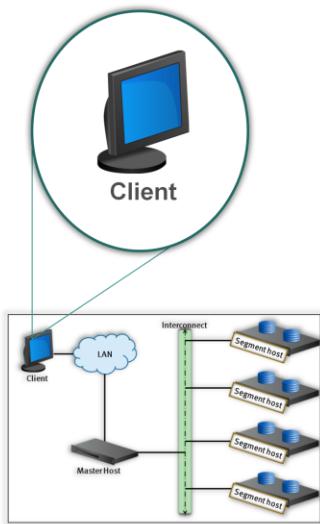
Greenplum Database also includes features designed to optimize Postgres for BI and DW such as external tables (parallel data loader), resource management, and query optimizer enhancements. Many BI features developed by Greenplum do make their way back into the PostgreSQL community. For example, table partitioning is a feature developed by Greenplum which is now in standard Postgres.



This slide introduces each primary or required component of a Greenplum DB system at a high-level. We will cover each component in detail on the next few slides.

- Users access a Greenplum database using various client applications and interfaces. All client interfaces supported by PostgreSQL are also supported by Greenplum Database.
- The access point to a Greenplum Database system is the master – clients always connect to the master instance. The master instance is the database process that accepts client connections and processes the SQL commands issued by the users of the system. The master does not contain any user data, storing only the system catalogs or metadata.
- The master coordinates the work among all of the segment instances in the system. Data resides in the segments. This is where the majority of the query processing takes place. User-defined tables and indexes are distributed across the available number of segments in the Greenplum system, each segment containing a distinct portion of the data.
- The Interconnect is the glue that brings all the distinct database instances together to act as one cohesive DBMS. The Interconnect component in Greenplum Database is responsible for moving data between the segments during query execution. The interconnect delivers messages, moves data, collects results, and coordinates work among the master and segments in the system. The Interconnect rides on top of a standard Gigabit Ethernet switching fabric over a (preferably) private local area network (LAN).

## Greenplum Architecture – Client Programs



Client programs include:

- psql
- pgAdmin 3
- ODBC drivers
- JDBC drivers
- Perl database interface (DBI)
- Python
- libpq

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

65

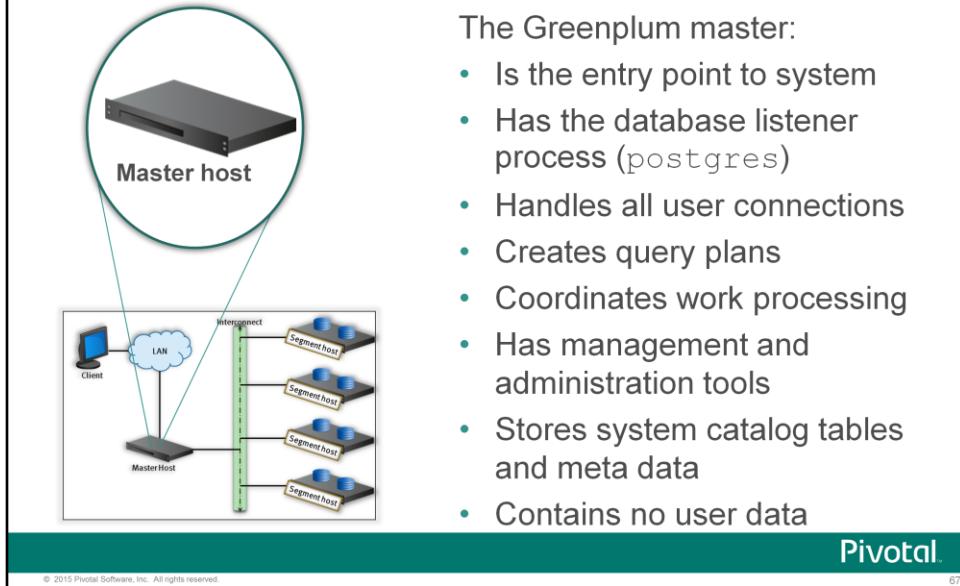
Greenplum Database uses the same client interfaces as PostgreSQL. Although there are other client interfaces available, the following client interfaces have been tested with Greenplum Database:

- **psql** – The command-line terminal interface to PostgreSQL, psql is bundled with your Greenplum installation. This terminal-based front-end command to PostgreSQL lets you access the PostgreSQL database interactively or through a file. It also provides meta-commands and other shell-like features to facilitate writing scripts and automating a variety of tasks.
- **pgAdmin 3** – pgAdmin is a graphical interface developed by an international team of PostgreSQL developers. It is a feature-rich, open-source administration and development platform. It contains a graphical interface to support all PostgreSQL features. It is free and available for download on many platforms.
- **ODBC** – Using the Postgres ODBC database driver, `psqlodbc`, Open Database Connectivity-compliant client applications, such as ETL or reporting tools, can be configured to connect to a Greenplum database.
- **JDBC** – The Postgres Java Database Connectivity (JDBC) database driver, `pgjdbc`, allows Java programs to connect to a PostgreSQL database using standard, database independent Java code. It provides reasonably complete implementation of the JDBC 3 specification in addition to some PostgreSQL specific extensions.

## **Greenplum Architecture – Client Programs (continued)**

- **PERL DBI** – PERL Database Interface is an API for connecting programs written in Perl to database management systems (DBMS). It is the most common database interface for the Perl programming language. The PostgreSQL Perl database driver, pgperl, can be used to access a Greenplum database.
- **Python** – There are several Python interfaces available for PostgreSQL. PyGreSQL is the one recommended on the postgresql.org web site.
- **libpq** – The PostgreSQL native C application programming interface to PostgreSQL, libpq, is a set of library functions that allow client programs to pass queries to a PostgreSQL backend server and to receive the results of these queries. libpq libraries are distributed with Greenplum Database.

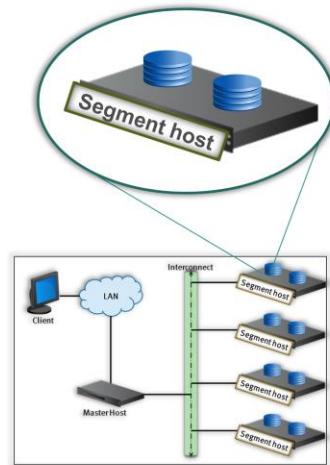
## Greenplum Architecture – Master Host and Instance



The Greenplum Database Master instance or master:

- Is the entry point to a Greenplum Database system. Users connect to the master and interact with a database as they would in any other DBMS. Any SQL statement or data load goes through the master, always.
- Has the PostgreSQL database listener process, `postgres`, also referred to as the *postmaster* in prior releases. This is the process to which users connect and initiate a database session. By default, it uses port 5432.
- Handles connections, authenticates users, and processes incoming requests.
- Creates query plan to distribute the query workload across all segment instances.
- Occasionally does some final processing for queries such as final aggregations, summations, orders and sorts. However, most of the workload of a query is handled on the segment instances.
- Is the entry point for all system administration tasks. All of the administration utilities and tools for Greenplum Database reside on the master installation.
- Does not contain any user data. User data is distributed on the segment instances. The only data contained on the master is the system catalog tables and system metadata.

## Greenplum Architecture – Segment Servers and Primary Segments



### Segments:

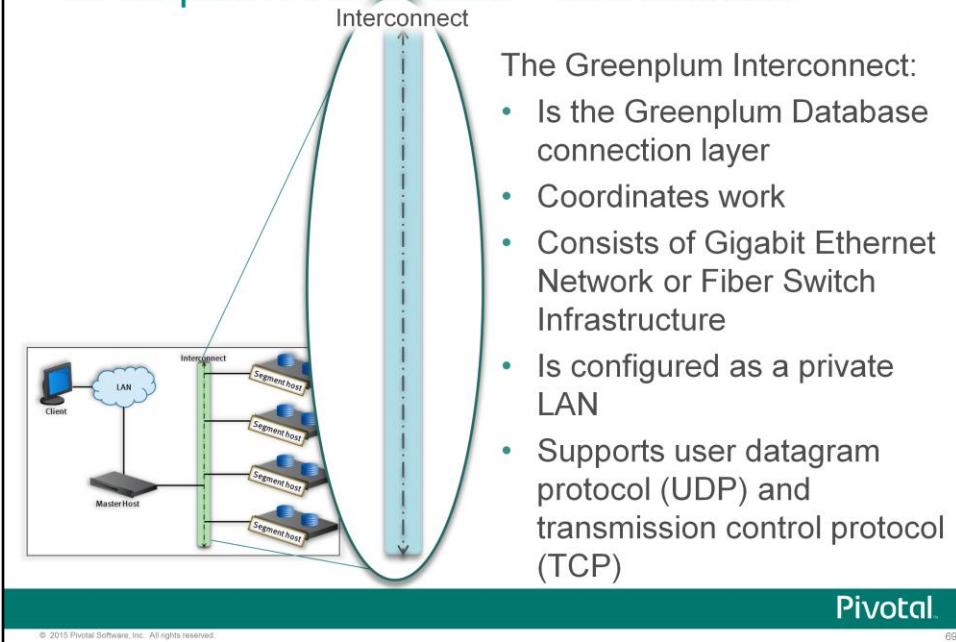
- Each contain a portion of user data
- Can have multiple per host
- Are not accessed directly by users – all connections go through the Master
- Have a segment listener process, `postgres`, which listens for connections from Master

Pivotal.

Segment servers and their segments have the following characteristics:

- All user-created tables and indexes are distributed across the segment instances or segments in the system. Each segment holds a portion of data for each distributed user table and index.
- A segment host can have multiple segment instances residing on it. Typically there is one primary segment instance per CPU core on a host in a production installation. If mirroring is enabled, there is a primary and mirror pair per CPU core.
- Users and administrators do not access the segments directly. All communications with the segments are through the master. While there is a way to connect to a segment in utility mode, the case for doing that is very rare and should not be done without talking to Greenplum support first.
- Each segment instance has a PostgreSQL segment listener process, `postgres`, that listens for connections from the master only. It will not accept client connections from database users. The segment port numbers are configurable at the time the system is initialized.

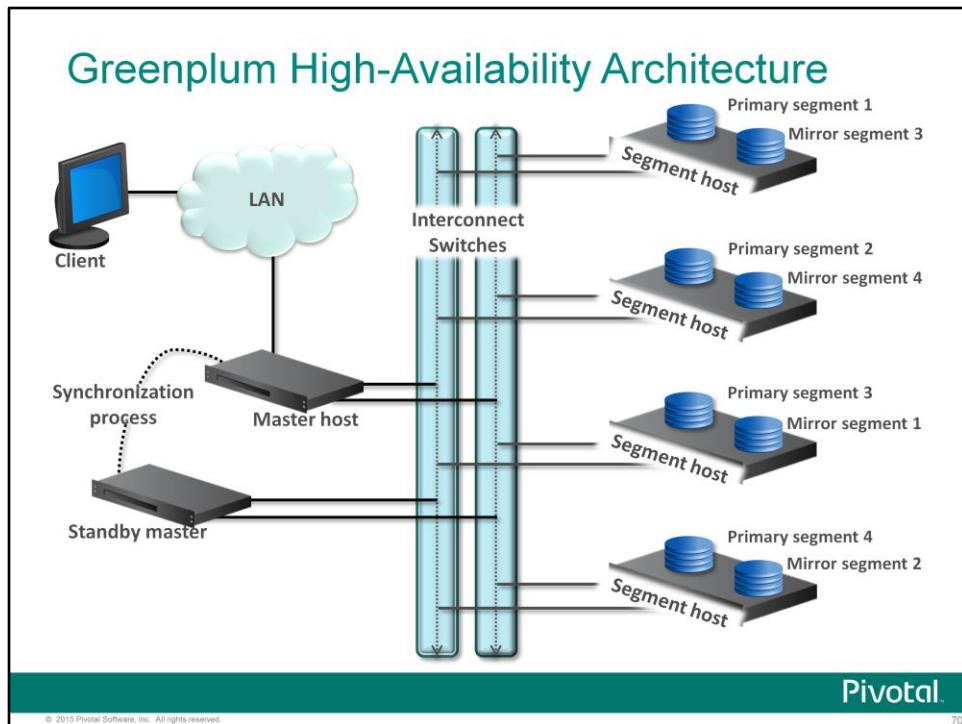
## Greenplum Architecture – Interconnect



The Interconnect component in Greenplum Database:

- Is the connection layer between all of the individual database instances, master and segments. It is the glue that holds all of the components together and makes the system act as one cohesive DBMS. This is the proprietary component developed by Greenplum that enables parallel processing.
- Is responsible for moving data between the segments during query execution. The interconnect delivers messages, moves data, collects results, and coordinates work among the segments in the system.
- Supports gigabit Ethernet network or fiber switch infrastructure as configured in the Pivotal Data Computing Appliance cluster.
- Is configured as a private LAN. Segment hosts should not be visible outside the Greenplum array.
- By default, the Interconnect uses user datagram protocol (UDP) to send messages over the network. The Greenplum software performs the additional packet verification and checking not performed by UDP, so reliability is equivalent to transmission control protocol (TCP). UDP performance is equivalent to or better than TCP.

**Note:** If performing data loads with ETL, the ETL server can be plugged directly to the interconnect so that it can communicate with the segment servers directly.

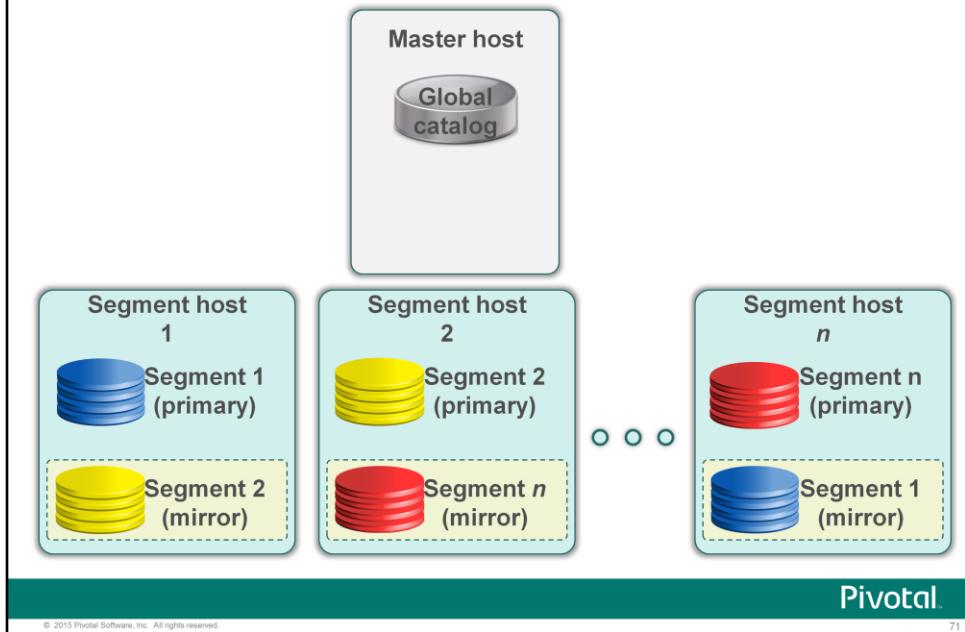


In addition to the primary Greenplum system components, you can also optionally deploy redundant components for high-availability and no single-point of failure.

For data redundancy, you can implement the following configuration:

- **Mirror segment instances** – A mirror segment always resides on a different host than its primary. Mirroring provides you with a replica of the database contained on a segment which may be useful if you lose a host in the array or a segment instance becomes corrupted, as may occur with disk failure.
- **Standby master host** – For a fully redundant Greenplum Database system, a mirror of the Greenplum master instance can be deployed. A backup Greenplum master host serves as a *warm standby* in the event that the primary Greenplum master host becomes inaccessible. The standby master host is kept up to date by a transaction log replication process that runs on the backup master host and keeps the data between the primary and standby master synchronized. Should the master host fail, log replication is shut down, the standby master is activated, and the transaction logs are used to reconstruct the state of the master at the time of the last successful transmission.
- **Dual interconnect switches** – A highly available Interconnect can be achieved by deploying dual Gigabit Ethernet or Fiber switches and redundant network interfaces on all the Greenplum Database hosts in the system. The default configuration is to have one network interface per primary segment instance on a segment host, each configured on its own subnet. If using dual switches, divide the subnets evenly between the switches.

## Data Redundancy – Segment Mirroring



In a Greenplum system, the master host stores only the system metadata. Each segment host has a distinct portion of the user data. With mirroring, a replica of a segment database resides on a different host. The mirror acts as an exact replica of the primary segment and is only activated should the primary segment fail. The mirror segment receives 32k pages of changes made to the primary.

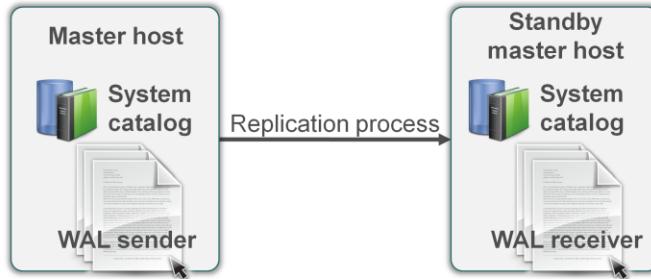
The following is an example of a redundant configuration:

- If the primary copy for Segment 1 is on segment host 1, the mirror copy is on another segment host, segment host  $n$ .
- The primary copy for Segment 2 is on segment host 2. Its mirror copy is on segment host 1.
- The primary copy for segment  $n$  is on segment host  $n$ . Its mirror copy is on segment host 2.

In this scenario, should you lose a segment host, you can continue to access all of the data segments as queries will fail over to the mirror segment should the primary segment be unavailable. Spread mirroring helps to balance the node so that should a segment fail, all of the remaining segments can distribute the load.

While the default mirror setup uses the same hosts as where your primaries are deployed, note that it is also possible to deploy your mirrors on a completely different set of hosts than your primaries, even hosts in a different physical location.

## Master Mirroring – Warm Standby



Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

72

Because the Master is the entry point to a Greenplum Database system, you may want to have a backup in case your primary master host becomes unavailable. You can do this by deploying a *warm standby master*. The standby master host is kept up to date by replication processes executing on both the master and standby servers. The `walsender` process executes on the master and streams committed transactions to the `walreceiver` on the standby server. This streaming replication is used to keep the standby master synchronized with the master server.

If the primary master fails, the log replication process is shutdown, and the backup master can be activated in its place.

Upon activation of the backup master, the replicated logs are used to reconstruct the state of the Greenplum master host at the time of the last successfully committed transaction.

Since the Greenplum master does not contain any user data, only the system catalog tables need to be synchronized between the primary and backup copies. These tables are not updated frequently, but when they are, changes are automatically copied over to the backup Greenplum master so that it is always kept current with the primary.

## Data Distribution and Parallel Query Execution

You will now examine the concepts:

- Data Distribution
- Parallel Query Execution

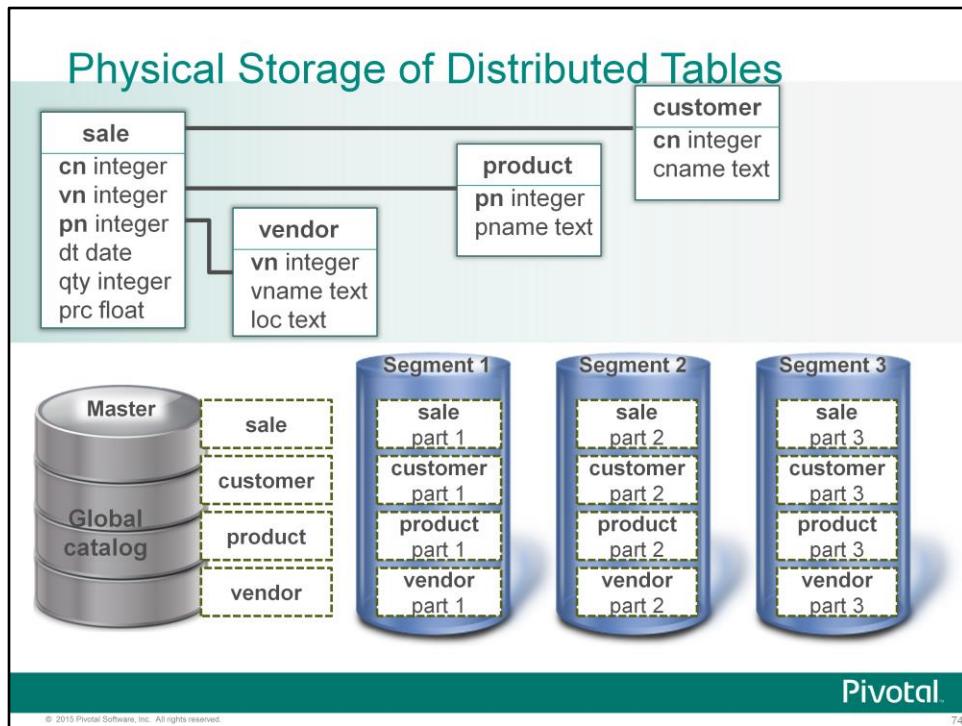
Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

73

In the next set of slides, you will examine key Greenplum Database concepts, including data distribution and parallel query execution.

Parallel query execution A lot of systems available in the marketplace today have parallel query execution. However, those databases do not support distributed tables. Greenplum is a true MPP system in that this is a shared-nothing architecture.



To understand how Greenplum Database stores data across the various hosts and segment instances, consider the following simple logical database schema.

This shows a simple star schema common in data warehousing. In this type of database schema, the **sale** table is usually called a fact table and the other tables (**customer**, **vendor**, **product**) are usually called the dimension tables.

Now let's see what these tables look like in the physical database.

In Greenplum Database, all tables are distributed. This means a table is divided into non-overlapping sets of rows or parts. Each part resides on a single database known as a segment within the Greenplum Database system. The parts are distributed evenly across all of the available segments using a sophisticated hashing algorithm.

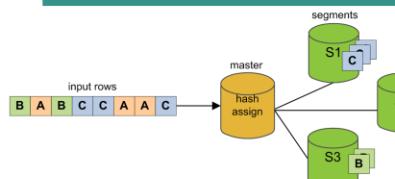
Distribution is determined at table create time by selecting a distribution key of one or more columns. Typically you would use a table's primary key or some other unique column or set of columns as the distribution key.

## Distribution Policies



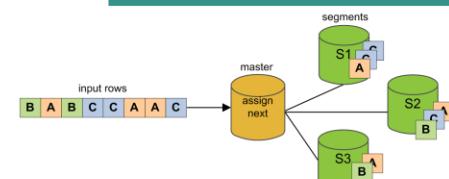
In hash distribution:

- Tables are created with the following syntax:  
CREATE TABLE ...  
DISTRIBUTED BY  
(column [...])
- Keys of the same value always sent to the same segments



In random distribution:

- Tables are created with the following syntax:  
CREATE TABLE ...  
DISTRIBUTED RANDOMLY
- Rows with columns of the same value are not necessarily on the same segment



Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

75

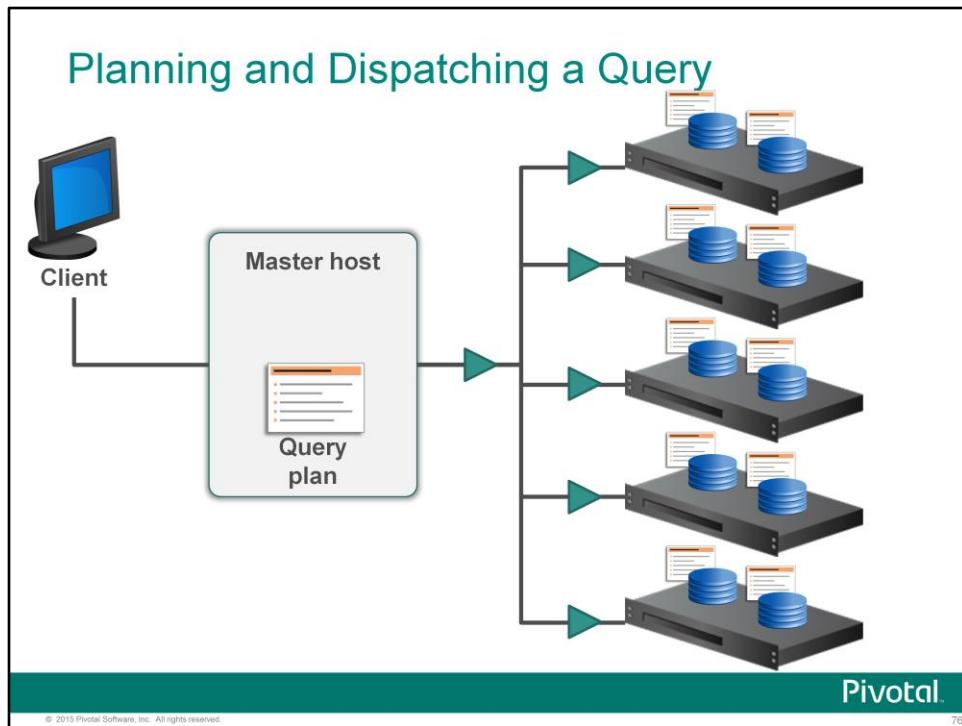
There are two distribution policies for dividing the rows among the available segments:

- **Hash** – In hash distribution, one or more table columns are used as the distribution key. These columns are used by a hashing algorithm to divide the data among all of the segments. The key value is hashed, or a random number created. This hashed value is divided by the number of segments. The remainder of this division is the assigned segment number. Keys of the same value will always hash to the same segment. Choosing a unique distribution key, such as the table's primary key, will ensure the most even data distribution.

**Note:** There are performance advantages to choosing a hash policy whenever possible. We will examine the performance implications of choosing one over the other later in the course.

- **Random** – In random distribution, a random number is used as the key to generate the hash. Once generated, the same behavior used for the hash distribution is also used here. The hashed value is divided by the number of segments and the mod is used to assign the row to the assigned segment number. While rows with the same value will not necessarily be assigned to the same segment, the assignment is more random but the distribution is within a 10% variance.

**Note:** External tables do not have a distribution policy associated with them as they are outside the realm of control for the Greenplum DB.



When working with Greenplum, you issue queries to the database as you would any other database management system (DBMS). You can connect to the database instance on the Greenplum master host using a client application, such as psql, and submit an SQL statement.

The master:

- Receives the query
- Parses the query
- Optimizes the query
- Creates a parallel query plan
- Dispatches the same plan to all of the segments for execution

Each segment is responsible for executing local database operations on its own set of data.

We will examine this concept in great detail later in the course.

## Lab: Greenplum Product Overview

In this lab, you review your knowledge on Greenplum Database concepts, architecture, and components.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

77

In this lab, you will review concepts you have learned in this lesson.

# Module 1: Greenplum Fundamental Concepts

## Lesson 4: Summary

During this lesson the following topics were covered:

- Primary architecture and components
- Redundant components
- Distributed data and queries

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

78

This lesson covered the primary architecture in greater depths, discussing the components found within the Greenplum environment, including the master, standby, and segment hosts, the redundant architecture to safeguard the environment and improve uptime, and how data is distributed and queries managed.

## Module 1: Summary

Key points covered in this module:

- Identified the basic elements and common methodologies employed in data warehousing
- Listed the features and benefits of implementing a Greenplum solution
- Highlighted key points of the shared-nothing, MPP design implemented in Greenplum
- Identify and describe the components of the Greenplum architecture and describe how Greenplum supports redundancy and high availability

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

79

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

- Define data warehouse, Big Data, and identify the basic elements and methodologies employed in data warehousing.
- List the features and benefits of implementing the Greenplum Database with applicable hardware solutions.
- Identify the key points of the shared-nothing, MPP design in Greenplum.
- Identify and describe the components of the Greenplum architecture and describe how Greenplum supports redundancy and high availability.

This slide is intentionally left blank.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

80

# Module 2: Database Installation and Initialization

This module presents reference architecture information and shows you how to install and configure your Greenplum database.

Upon completion of this module, you should be able to:

- Use available verification tools to verify optimal performance for the Greenplum database
- Initialize and validate a Greenplum installation

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

1

In this module, you install and initialize the Greenplum Database and Greenplum Database instance. You will configure the database and begin the process of loading data.

You will install Greenplum and use the available verification tools to ensure that the environment has been properly configured to optimize data movement, storage, and retrieval. Once verification is completed, a Greenplum Database instance can be installed and initialized to prepare it for use.

## Module 2: Database Installation and Initialization

### Lesson 1: Systems Preparation and Verification

In this lesson, you will examine reference architecture for Greenplum and the steps required to prepare a system prior to installing Greenplum.

Upon completion of this lesson, you should be able to:

- Identify Greenplum software and hardware solutions and requirements
- Describe reference architecture for Greenplum
- List verification steps and tools to prepare a system for Greenplum

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

2

In this lesson, you will examine the supported operating systems and EMC hardware available for your Greenplum environment. You will also examine reference architecture for Greenplum, and list the steps and tools you can use to prepare a system for Greenplum installation and initialization.

## Greenplum Hardware Considerations

The Greenplum software-only solution:

- Gives customers a choice of hardware platforms
- Is dependent on hardware solutions to optimize performance
- Performs well on certified reference architecture

The following statements are true for general hardware configuration:

- Segment servers should have identical hardware specifications
- Master server requires fast CPU and lots of RAM

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

3

Greenplum Database is a software only solution, meaning that can theoretically run on any hardware platform. The hardware and software are not coupled as in appliance vendors like Netezza and Terradata. However, as with any database, Greenplum's performance is dependent on the hardware on which it is installed. As the database is distributed across multiple machines in a Greenplum Database system, the selection and configuration of hardware is even more important to achieve the best performance possible from your database. Using SSD disk drives, for example, can greatly improve the performance of the database from the recommended specifications provided for Greenplum DB.

Greenplum provides reference architecture that you can use in building a white-box solution.

In general, when configuring the hardware environment for your Greenplum software-only solution, remember that:

- Segment servers should have identical hardware specifications. As they work in parallel, it is important that the components of the architecture are the same to provide optimal performance and throughput.
- The master server needs fast CPU and lots of RAM to handle incoming queries, create the query plans, and distribute these out to the segment servers.

We will examine the recommended hardware configurations for:

- The segment hosts, which store the user data and do the majority of the data processing.
- The master host, which handles user connections.
- Interconnect, which is the data transport layer between the hosts in the array.

# Greenplum Database 4.3.x.x System Requirements

<b>Operating System</b>	<ul style="list-style-type: none"><li>SuSE Linux Enterprise Server 64-bit 10 SP4, 11 SP1, 11 SP2</li><li>CentOS 64-bit 5.0 or higher</li><li>RedHat Enterprise Linux 64-bit 5.0 or higher</li><li>Oracle Unbreakable Linux 64-bit 5.5</li><li>Mac OSX 10.5 or higher (<i>Greenplum Community Edition – single node edition</i>)</li></ul>
<b>File System</b>	xfs required for data storage on SUSE Linux and Red Hat (ext3 supported for root file system)
<b>Minimum CPU</b>	Pentium PRO compatible (P3/Athlon and above)
<b>Memory</b>	16 GB RAM per server (minimum) 64+ GB RAM per server (recommended)
<b>Disk Requirements</b>	<ul style="list-style-type: none"><li>150 MB per host for Greenplum installation</li><li>Approximately 300 MB per segment instance for meta data</li><li>Appropriate free space for data with disks at no more than 70% capacity</li><li>High-speed, local storage</li></ul>
<b>Network Requirements</b>	<ul style="list-style-type: none"><li>Gigabit Ethernet within the array</li><li>Dedicated, non-blocking switch</li></ul>

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

4

The minimum recommended system specifications for installing Greenplum Database 4.3 are as shown on the slide. Greenplum is supported on major UNIX-based systems.

For each server in the array, the minimum memory is 16 GB of RAM. 64 GB of RAM or more is recommended for all servers. The master and standby server may require additional memory if other tools or databases are included, including GPTEXT and Pivotal HD.

Disk requirements vary based on your needs, but a minimum 150MBytes for the installation and 300MBytes per segment for meta data is required. The disk capacity should be no more than 70% to maintain optimal performance. We will discuss estimating disk capacity for user data later in the course.

EMC provides a free Greenplum Database software called Greenplum Database Community Edition to be used in single node test environments. This single node edition shares much of the same requirements as the production software. It differs in the following areas:

- It can be installed on Mac OSX 10.5 or higher. This platform is not supported for production installation of Greenplum Database.
- Minimum memory requirements are 1GB. However, for optimal performance, 4GB per CPU core is recommended.
- It does not share the same networking requirements as the multi-node production database software.
- xfs is recommended for optimal performance, but is not required. Either xfs or ext3 can be used for file system support.

Note: Starting with Greenplum Database 4.3.0.0, Solaris is no longer a supported operating system.

## Estimating Storage

Total raw disk capacity required must take into account the following:

Type	Amount of Storage to Allocate
RAID parity/mirroring	Depends on RAID type chosen; i.e., 50% of storage for RAID 10
Greenplum Database segment mirrors	50% of storage
File system overhead	About 10% of storage
Used capacity	Less than 70% recommended
Raw data size and database storage overhead	Raw data may be 1.4 times larger on disk; depends on table types, indexes, compressions, etc.
System metadata	About 20 MB per segment and master instance
Write ahead log (WAL)	About 1088 MB per segment and master instance
Greenplum Database log files	About 10 MB per segment and master instance

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

5

To estimate your storage needs, you need to calculate the raw disk capacity available for data storage on all of your segment hosts and consider the following:

- **RAID Parity and Mirroring** – Your overall disk space capacity will be reduced according to the RAID level you choose. For example, if using RAID 5, subtract one disk drive's worth of storage capacity per each RAID 5 set. If using RAID 10, cut your total disk drives' storage capacity in half.
- **Greenplum Database Mirroring** – Doubles the storage requirements of your user data on segment hosts.
- **Less than 70% Capacity** – For optimal performance, disks should not be more than 70% full. Performance drops off precipitously after 70%. If you do start hitting 70%, you should look at options including extending the appliance or hardware.
- **File system overhead** – The file system overhead can be anywhere from 5-15% (checksumming, file permissions, etc) depending on the file system you are using.
- **Raw Data Size and DB overhead** – As with all databases, the size of your raw data will be slightly larger once it is loaded into the database. On average raw data will be about 1.4 times larger on disk after it is loaded into the database, but could be smaller or larger depending on the data types you are using, size of the rows versus page size, whether you create indexes, etc. The installation guide has detailed sizing info if you want to calculate DB overhead for your data more accurately.

## Estimating Storage (continued)

- **System Metadata** – For each Greenplum Database segment instance (primary or mirror) or master instance running on a host, estimate approximately 20 MB for the system catalogs and metadata.
- **Write Ahead Log** – For each Greenplum Database segment (primary or mirror) or master instance running on a host, allocate space for the write ahead log (WAL). The WAL is divided into segment files of 64 MB each. At most, the number of WAL files will be:  $2 * \text{checkpoint segments} + 1$ . You can use this to estimate space requirements for WAL. The default *checkpoint segments* setting for a Greenplum Database instance is 8, meaning 1088 MB WAL space allocated for each segment or master instance on a host.
- **Greenplum Database Log Files** – Each segment instance and the master instance generates database log files, which will grow over time. Sufficient space should be allocated for these log files, and some type of log rotation facility should be used to ensure that the log files do not grow too large.
- **Performance Monitor Data** – Greenplum Command Center agents run on the same set of hosts as your Greenplum Database instance and utilize the system resources of those hosts. The resource consumption of Greenplum Command Center agent processes on these hosts is minimal and should not significantly impact database performance. Historical data collected by Greenplum Command Center monitoring data is stored in its own `gpperfmon` database within your Greenplum Database system. Collected monitor data is distributed just like regular database data, so you will need to account for disk space in the data directory locations of your Greenplum segment instances. The amount of space required depends on the amount of historical data you would like to keep.

The next slide shows the calculations used for determining the disk space requirements. You will still need to account for the user data size, based on the type of data, storage, and compression methods used to name a few. You will also still need to account for system logs and metadata.

## Calculating Usable Disk Capacity

### ① Calculate raw capacity across all segments

```
raw_capacity = disk_size * number_of_disks
```

### ② Calculate formatted disk space with appropriate overhead and RAID taken into account

```
formatted_disk_space = (raw_capacity * .9) / 2
```

### ③ Calculate usable disk space (less than 70 %)

```
usable_disk_space = formatted_disk_space * 0.7
```

### ④ Compare against user data (U) and work area (1/3U) with and without mirrors

Without mirror:  $U + U/3$

With mirror:  $2U + U/3$

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

7

Taking into account all of the afore mentioned items you must consider when calculating required storage size, here are some measurements that will guide you in calculating how much storage you will require for your environment.

Usable disk space is calculated first for each segment host, taking into account RAID configurations, and finally mirroring. To do this:

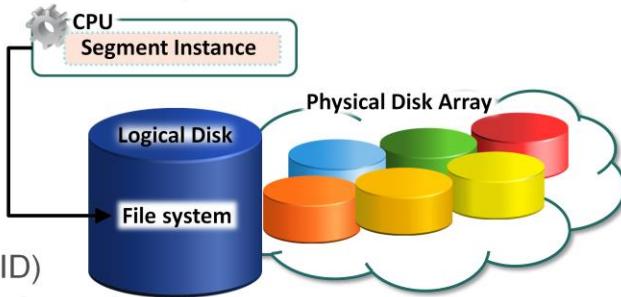
- First, calculate the raw capacity for the entire system by multiplying the number of available disks across the entire cluster by the disk size. The calculation is  
$$\text{raw\_capacity} = \text{disk\_size} * \text{number\_of\_disks}$$
- Next, account for both RAID and any file system overhead, which may on average be 10%. The formatted disk space would be calculated as follows if RAID 10 is being used with 10% file system overhead:  
$$\text{formatted\_disk\_space} = (\text{raw\_capacity} * .9) / 2$$

Remember, RAID 10 consumes half of the available disk drive. Use the appropriate calculation for your implementation of RAID.
- Ensure you are not using more than 70% of the disk drive space to maintain optimal disk drive performance:  
$$\text{usable\_disk\_space} = \text{formatted\_disk\_space} * .70$$
- Once you have calculated the usable disk space, examine the user data size, represented by  $U$ , to ensure it corresponds to the usable disk space calculated here. If you are using mirrors, you will require twice the disk space ( $U * 2$ ). Additionally, a work area for active queries needs to be set aside. The work area should be 1/3 the size of the user data size, calculated as  $U/3$ .

## Segment Host – Disk Layout

An optimal disk layout includes:

- One primary segment instance per effective CPU
- Primary segment mapped to a file system within a logical disk drive
- Logical drive uses groups of physical disks (RAID)
- RAID level chosen depends on:
  - Performance versus capacity requirements (RAID-10 or RAID-5)
  - Data protection and disk fault tolerance requirements



Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

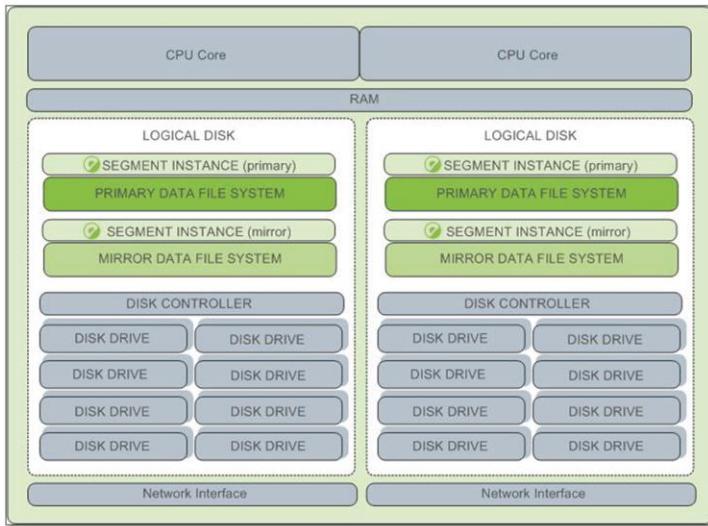
8

An optimal disk layout for Greenplum Database maps a single primary Segment Instance to a filesystem running within a logical disk drive. The logical disk and filesystem are provided by the operating system, and can be chosen from many available options. Most operating systems provide the ability for a logical disk drive to use groups of physical disks arranged in RAID arrays.

The RAID level you choose should be chosen based on:

- **Performance over capacity** – For example, a RAID 10 (mirrored) disk configuration offers the best performance while a RAID 5 or RAID 1 configuration has about 30% slower performance but more capacity.
- **Data protection** – RAID 10 reserves half of the disks for data parity while RAID 5 reserves one disk for data parity.

## Segment Host Configuration



Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

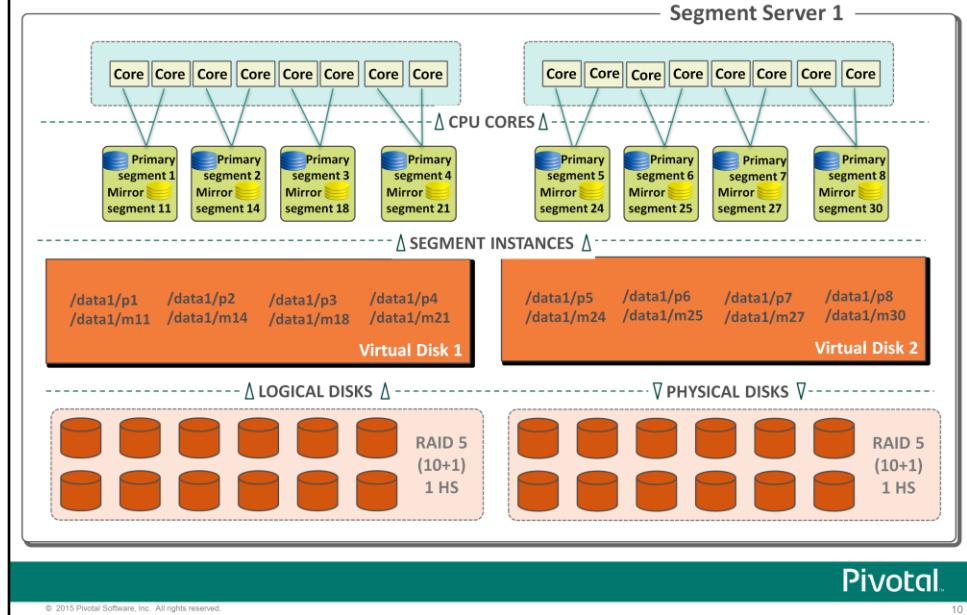
9

This is an example of the hardware stack configuration of a segment host:

- One PRIMARY Greenplum Segment Instance per CPU core (or primary/mirror pair)
- A minimum of 64GB for a production environment to 256GB of RAM, depending on additional tool usage or production requirements
- One logical disk drive per CPU with 2 file systems – one for primary segments and one for mirror segments
- Hardware or software RAID disk controllers (one per set of drives)
- A number of disk drives divided evenly across primary segments
- One 10 GigE Network Interface per primary segment instance where each NIC is on its own subnet to balance Interconnect traffic

The number of effective CPUs on a host is the basis for determining how many primary Greenplum Database segment instances to deploy per segment host. This example shows a host with two effective CPUs (one dual-core CPU). Note that there is one primary segment instance, or primary/mirror pair if using mirroring, per CPU core.

## Anatomy of a Segment Server



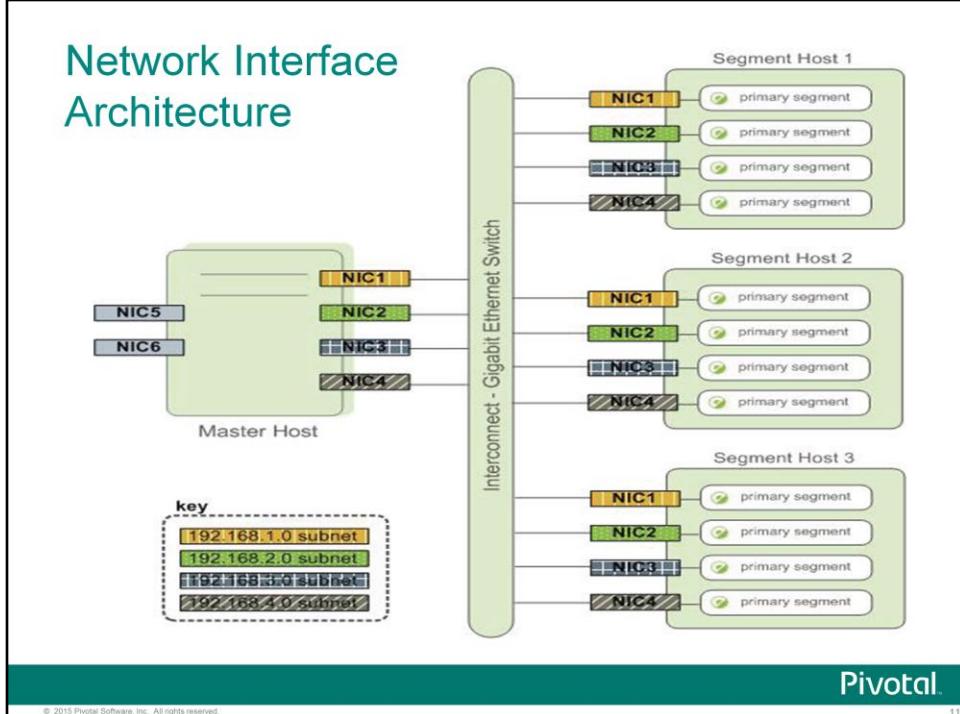
The illustration on the slide shows the segment server configuration in the Pivotal DCA. Each primary segment instance and mirror segment instance pair has the use of two CPU cores. The minimum that can be configured is a primary-mirror pair to a single CPU core. If you are using compression in your tables, increasing the number of cores per pair gives you better CPU cycles for compression and decompression.

The mirrors shown in this illustration will pair with a primary segment on a different segment host or node. For example, primary segment instance 1 pairs with mirror segment instance 24. By keeping the mirrors on a separate node, should the primary segment become unavailable on one node, the mirror segment from another node will activate.

Each server has two CPU processors with 6 CPU cores each for a total of 12 available CPU cores. Each segment server also supports 12 disk drives where the drives are divided into two groups of 6 disk drives with RAID 5 (5+1). Each group has two volumes with the following configuration:

- Volume disk 1 in the first group contains the `/root` partition.
- Volume disk 2 in the first group contains the `/data1` partition.
- Volume disk 1 in the second group contains the `/swap` partition.
- Volume disk 2 in the second group contains the `/data2` partition.

# Network Interface Architecture



A segment host typically has one network interface per primary segment instance. If using mirroring, a primary/mirror pair would then share an interface. The master host would also have four network interfaces to the Greenplum Database array plus additional external network interfaces.

On each Greenplum Database segment host, you would then create separate host names for each network interface. For example, if a host has four network interfaces, then it would have four corresponding host names, each of which will map to a primary segment instance. You would also do the same for the master host, however, when you initialize your Greenplum Database array, only one master host name will be used within the array.

With this configuration, the operating system automatically selects the best path to the destination. Greenplum Database automatically balances the network destinations to maximize parallelism.

## Pivotal DCA – Pivotal Master and Segment Node

	Pivotal Compute	Pivotal Hi-Memory	Pivotal Standard			
System	Intel (16 CPU Cores)					
Memory	64 GByte	256 GByte	64 GByte			
Disk drives	<ul style="list-style-type: none"><li>• 1 x Dual channel 6GB/s SAS internal RAID card</li><li>• 6 x 300 GByte SAS (master/standby)</li></ul> <table><tr><td>24 x 300 GByte SAS 10K (segment servers)</td><td>24 x 300 GByte SAS 10K (segment servers)</td><td>24 x 900 GByte SAS 10K (segment servers)</td></tr></table>	24 x 300 GByte SAS 10K (segment servers)	24 x 300 GByte SAS 10K (segment servers)	24 x 900 GByte SAS 10K (segment servers)		
24 x 300 GByte SAS 10K (segment servers)	24 x 300 GByte SAS 10K (segment servers)	24 x 900 GByte SAS 10K (segment servers)				
Network interfaces	<ul style="list-style-type: none"><li>• 4 x 1Gb/s network interfaces (master/standby)</li><li>• 2 x 1Gb/s network interfaces (segments)</li></ul>					
Administrative interface card	1 x Intel BMC					
Operating system	Redhat Enterprise Linux distribution x86 32/64-bit license					
Network adapter	2 x dual-port 10Gb Converged Network Adapter (CNA)					

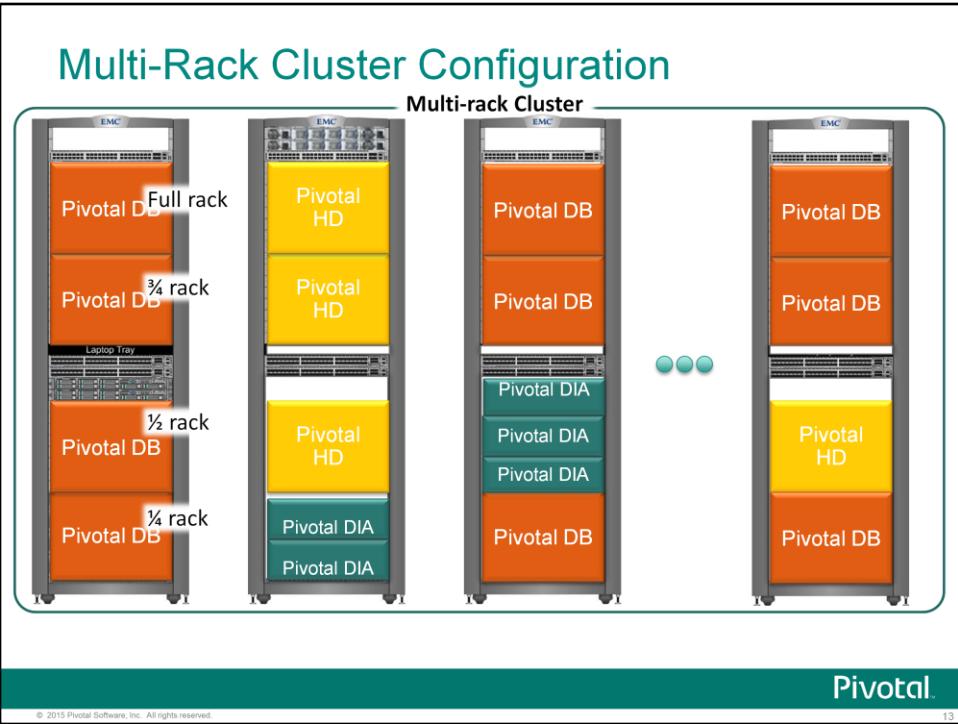
Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

12

Now that you have examined the hardware requirements and overall architecture should you decide to build your own hardware solution, we will examine the solutions provided by EMC and Pivotal.

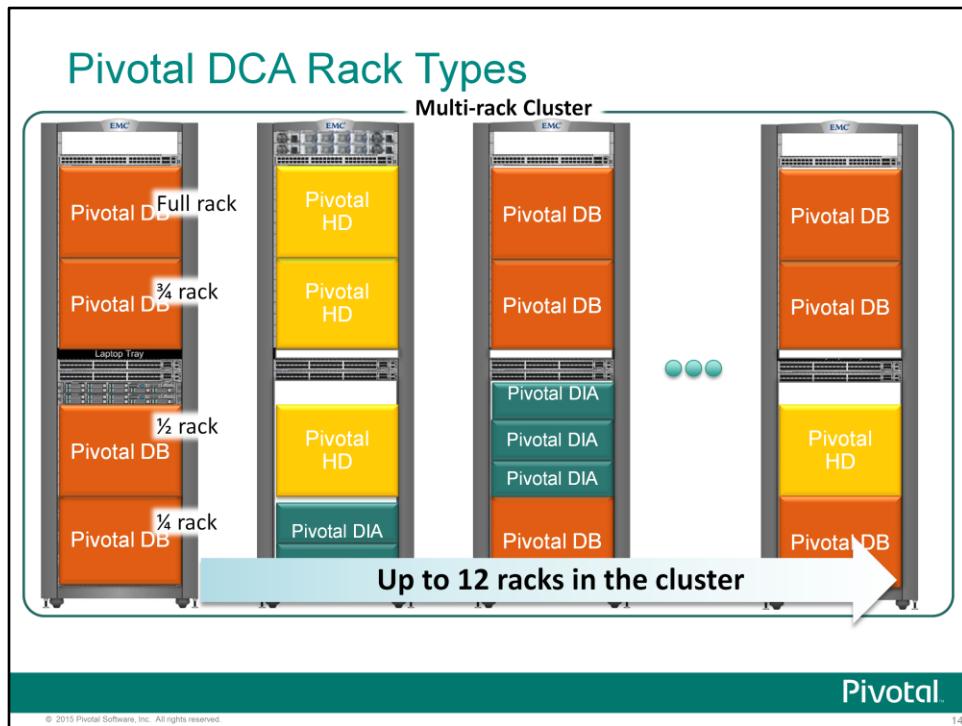
The slide highlights the hardware specifications for the Pivotal DCA. The Pivotal DCA uses Intel servers for all servers within a module. Disk capacity and memory may vary depending on the module type.



The Pivotal DCA is a self-contained data warehouse solution that integrates all the database software, servers and switches necessary to perform enterprise-scale data analytics workloads. The Pivotal DCA is delivered racked and ready for immediate data loading and query execution.

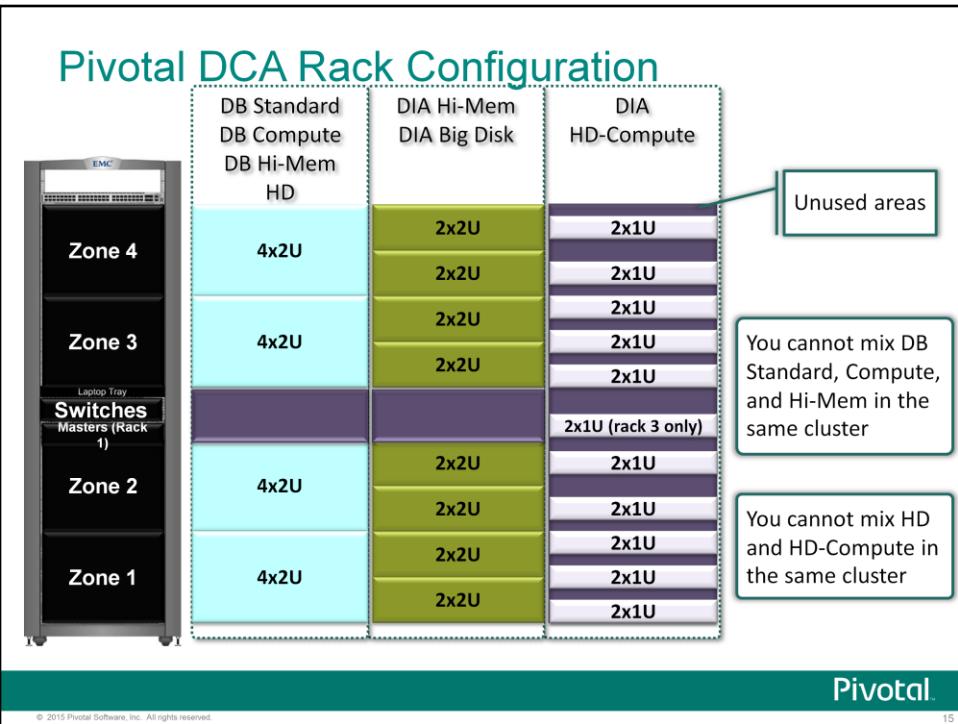
The Pivotal DCA is offered in increments of four servers, starting with a quarter-rack configuration to a multi-rack configuration. The base architecture of DCA is designed with scalability and growth in mind. This enables organizations to easily extend their DW/BI capability in a modular way; linear gains in capacity and performance are achieved by expanding a DCA system to take advantage of additional processing power and memory.

The cluster can be populated with Pivotal DB standard or compute modules, but not both. Within the cluster, you can also support up to 11 DIA modules, and Pivotal HD or Pivotal HD-compute modules.



In a multi-rack configuration, you will find:

- The base or standard rack which contains the master servers for the entire cluster.
- An aggregate rack which is used to provide inter-rack communication for all of the racks of the cluster. This rack contains two additional racks to accommodate communication among all racks within the cluster.
- One or more expansion racks, depending on the number of modules you have installed, which will expand the cluster up to, and including, twelve racks. This is dependent on the number of ports on the aggregate switch supplied in the cluster.

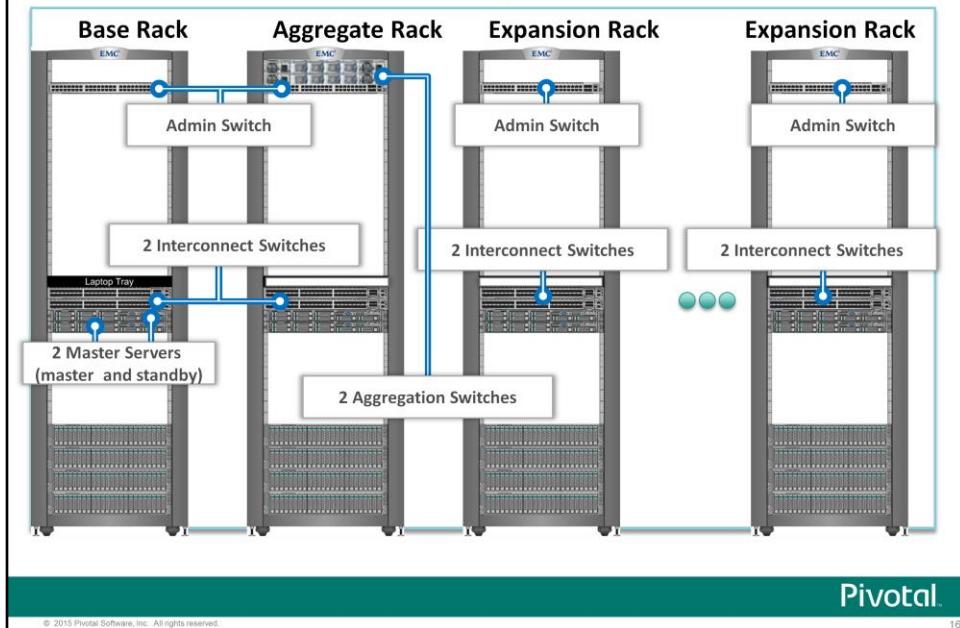


The Pivotal racks provide up to 40 rack units of usable space and supports a variety of configurations. Each zone, as defined supported by rack power, is eight rack units high.

High memory modules reduce the number of servers that can be placed in any one rack, and in the end, in the total cluster overall. You can mix database modules with DIA modules and HD modules. You cannot combine multiple types of database modules or multiple types of HD modules in the cluster however – you must choose one of those modules.

High memory modules impose limits on the number of modules that can be installed within a rack. Zone 4 cannot be populated if you are using high memory modules.

## Pivotal DCA Rack Components



The Pivotal DCA for Greenplum DB is shipped with:

- Greenplum Database software.
- Two master servers in the base, where one acts as the primary master server and the other acts as the hot standby master server.
- Segment Servers to contain the data and handle the bulk of the query processing. The number of segment servers varies based on the number of Greenplum Database modules installed. At a minimum, each segment server has six primary segments and six mirror segments for a total of 12 segments. A fully populated rack with four modules, each with four segment servers, will display 192 segments.
- High-speed dual interconnect switches to handle requests from the master to the segments, between segments, and to provide high-speed access to the segment servers for data loading.
- An admin switch to provide access to an administration network that is used to manage the DCA system components. The admin switch consists of one 48-port 1 GB Ethernet Layer 3 switch. It is used to connect network-enabled management ports from devices in the cluster. This switch can be cross-connected to a customer switch to provide console access to the cluster from a customer network.
- Aggregate switches in the aggregate rack to support communication between all segment servers in all of the racks of a multi-rack cluster.
- Note: Admin switches are the Arista 7048T switch on the V2 DCAs
- Note: Interconnect and aggregate switches are the Arista 7050S on the V2 DCA

Sizing a Pivotal DCA for Greenplum Database – Performance and Capacity Considerations				
	DB Standard Module		DB Compute/High-Memory Module	
	Quarter-Rack	Full Rack	Quarter-Rack	Full Rack
Master Servers	2		2	
Segment Servers	4	16	4	16
Total CPU Core	64	256	64	256
Total Memory	256 GB	1024 GB	256 GB / 1024 GB	1024 GB / 4096 GB
Segment HDDs	96	384	96	384
Usable Capacity Uncompressed	27.5 TB	110 TB	9 TB	36 TB
Usable Capacity (compressed)	110 TB	440 TB	36 TB	144 TB
Scan Rate	10 GB/Sec	40 GB/Sec	10 GB/Sec	40 GB/Sec
Data Load Rate	4 TB/Hour	16 TB/Hour	4 TB/Hour	16 TB/Hour

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

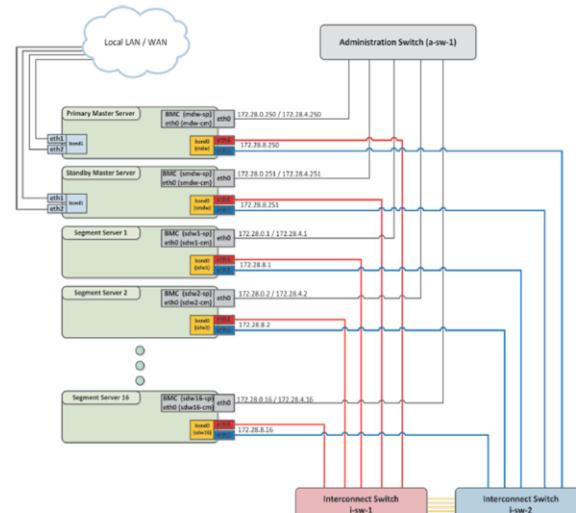
17

The tables on this and the next slide highlight the configurations for the standard DCA and high capacity DCA. Your needs will dictate the configuration that is right for your DW/BI needs.

The Pivotal Database Standard modules provide additional support for data-intensive applications by offering large disk drives at 27.5 terabytes for a quarter-rack configuration. The Pivotal Database Compute modules offer a lower-price point where the larger disk size is not required.

Note that the usable capacity on disks is based on RAID-5 configuration.

## Pivotal DCA Network Configuration



In this private LAN configuration, there are:

- Two interconnect networks
- Direct connections from all segments to both interconnects
- Admin switch for out-of-band management

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

18

The diagram shows an example of how the network is configured in a fully-populated Pivotal DCA base rack. The Greenplum Database interconnect and administration networks are configured on a private LAN. Outside access to Greenplum Database and to the DCA systems goes through the master host.

To maximize throughput, interconnect activity is load-balanced over two interconnect networks. To ensure redundancy, a primary segment and its corresponding mirror segment utilize different interconnect networks. With this configuration, Greenplum Database can continue its operations in the event of a single interconnect switch failure.

## Configuring the System for Greenplum

To prepare the Greenplum environment, you must:

1. Verify the system meets the base system requirements
2. Tune the kernel for your operating system
3. Install the Greenplum binaries on the master and segments and create the Greenplum administrative user
4. Perform hardware verification tests

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

19

Once you have determined the hardware needs for your Greenplum environment, you will continue configuring the environment for the Greenplum database. Specifically, you will:

1. Verify that the system meets the base requirements defined.
2. Tune the kernel parameters for the operating system you will be using in the Greenplum environment.
3. Run the Greenplum software installer to install the binaries on the master and segments. You will also need to create a Greenplum administrative user to handle the database initialization and configuration.
4. Validate the system configuration by performing hardware stress and verification tests.

## Linux Operating System Kernel Tuning

Shared Memory Kernel Parameters	Networking Kernel Parameters
kernel.shmmmax = 5000000000	net.ipv4.tcp_syncookies = 1
kernel.shmmni = 4096	net.ipv4.ip_forward = 0
kernel.shmall = 4000000000	net.ipv4.conf.default.accept_source_route = 0
kernel.sem = 250 512000 100 2048	net.ipv4.tcp_tw_recycle=1
kernel.sysrq = 1	net.ipv4.tcp_max_syn_backlog=4096
kernel.core_uses_pid = 1	net.ipv4.conf.all.arp_filter = 1
kernel.msgmnb = 65536	net.ipv4.ip_local_port_range = 1025 65535
kernel.msgmax = 65536	net.core.netdev_max_backlog=10000
kernel.msgmni = 2048	net.core.rmem_max = 2097152
vm.overcommit_memory=2	net.core.wmem_max = 2097152

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

20

The kernel must be tuned before proceeding with the Greenplum installation. If you are installing in a single virtual environment or physical machine, you will only need to configure a subset of the parameters shown. Otherwise, the following must be configured:

- **Shared Memory** – A segment or master instance (or any PostgreSQL database) will not start unless the shared memory segment for your kernel is properly sized. Most default Linux installations have the shared memory value set too low. Add the following lines to the `/etc/sysctl.conf` file of the master and all segment hosts.
- **Disable OOM killer** – OOM (out of memory) killer exists because the Linux kernel, by default, can commit to supplying more memory than it can actually provide. When the size of the data to be copied exceeds the size of physical memory, OOM killer randomly begins killing processes in order to free memory, often with bad results to any running Greenplum Database queries. In Greenplum Database, a single SQL statement creates several processes on the segment hosts to handle the query processing. Large queries will often trigger the OOM killer (if it is enabled) causing the query to fail. By changing the `vm.overcommit_memory` to 2, you disable the OOM killer.

## Linux Operating System Kernel Tuning (Cont)

User Limits (Defined in /etc/security/limits.conf)	XFS Mount Options
* soft nofile 65536	rw, noatime, inode64, allocsize=16m
* hard nofile 65536	Open files set to a minimum of 65536
* soft nproc 131072	
* hard nproc 131072	Max user processes set to a minimum of 131072
Block Device Options	Value
I/O Scheduler	deadline
Block device read-ahead value	16385

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

21

- Network** – The settings in /etc/sysctl.conf prevent Interconnect errors on large volume Greenplum systems. They allow for more efficient connection handling and larger volumes of connections required for distributing and executing extremely large query plans.
  - User Limits** – User limits control the resources available to processes started by a user's shell. Use ulimit, a system command to see the user limits for the currently logged in user. For Greenplum Database, each OS user that issues SQL statements needs to have their limits for open file descriptors and maximum processes increased.
  - Disk and mount options** – As with other databases, the Greenplum Database is expecting certain behaviors from the block devices and file systems. The xfs file system should be mounted with the options, rw (read and write access), noatime (do not update the file access times on the device), inode64 (use 64-bits for inodes), and allocsize=16m (increase the preallocation size to accommodate large files).  
The I/O scheduler and read-ahead values for block devices must also be updated.  
Reboot after making changes to the kernel.
- Refer to the Greenplum Database Installation Guide for system requirements for Solaris and MacOS-based systems.

## Disk Device and OS Settings

1

Update mount options for XFS file systems

Mount  
options

Set mount option to: `rw, noatime, inode64, allocsize=16m`

2

Set the I/O scheduler to `deadline` for all devices

I/O  
Scheduler

Use the following command on each device:

```
echo deadline > /sys/block/device_name/queue/scheduler
```

3

Change the read-ahead value for each block device to 16385

Read-  
ahead

Use the following command on each block device:

```
/sbin/blockdev --setra 16385 /dev/block_device_name
```

4

Disable Transparent Huge Pages (THP). (RedHat Linux 6.0 and higher)

THP degrades Greenplum Database performance.

Use the following command on to disable THP:

```
echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

22

In addition to ensuring kernel parameters are configured with appropriate values, you should also update the file system mount parameters, disk access policies, and disk read-ahead values for Greenplum to improve performance overall for reading and writing to disks.

If you are using the XFS file system for data storage, update the mount options for the file system to include the following: `rw, noatime, inode64, allocsize=16m`. This sets the file system with read and write access, no access time updates, use 64 bits when allocating inodes and data, and increase the preallocation size to 16 MB to ensure large files have some continuous space set aside for writes.

For the I/O scheduler, Greenplum, as do other databases, recommends using `deadline` for the I/O scheduler. This improves performance and allows all I/O requests to be processed within a specified period of time. This must be done for each block device using the following command: `echo deadline > /sys/block/device_name/queue/scheduler`, where the `device_name` could be `sdb`.

Finally, change the read-ahead value each block device to 16385, using the following command: `/sbin/blockdev --setra 16385 /dev/block_device_name`, where `block_device_name` could be `sdb`.

Disable Transparent Huge Pages (THP). RedHat Enterprise Linux 6.0 or higher enables THP by default. THP degrades Greenplum Database performance.

# Greenplum Database Installation Overview

1

Install the Greenplum Database binaries on the master server



- a. Download the Greenplum Database binary
- b. Unzip and execute the installation program as `root`

2

Install the Greenplum Database binaries on the standby and segment servers



- a. Access the master server as `root` and source `/usr/local/greenplum-db/greenplum_path.sh`
- b. Verify that the `/etc/hosts` file on all systems have the correct host names
- c. Create an exchange key list file with the hostname of each segment interface, master interface and standby interface.
- d. Run the `gpseginstall` utility to install the Greenplum Database binaries on the standby and segment servers

3

Verify the installation was successful



- a. Log in as the `gpadmin` user and source `/usr/local/greenplum-db/greenplum_path.sh`
- b. Run a command on all hosts using `gpssh` and verify you are not prompted for a password

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

23

When you begin to install Greenplum software, you always start with the master server. To install the Greenplum Database:

1. Download the Greenplum installer package and install the binaries on the master server. This will install the entire Greenplum DB package including the database server binaries, client programs, documentation, demo programs, management utilities, data loading binaries, bundled programs, and share files.
2. A `greenplum_path.sh` file is created by the installer and has the required settings needed for GPDB. As `root`, source this file after logging into the master server. You will then create an exchange key list that contains the host name of each interface for each segment instance as well as the standby server on a separate line. This file will be used to perform the next step, where you execute `gpseginstall`, to install the binaries on all instances identified in the key exchange file and create the Greenplum system administration user, `gpadmin`, with the password of your choosing. The `gpadmin` user is used for system administrative tasks as well as to run Greenplum. It cannot and should never be run as `root`.
3. Once the installation has completed, log into the master server as the `gpadmin` user and source the `/usr/local/greenplum-db/greenplum_path.sh` file. You can source this in `/etc/profile` or your Greenplum system administrator's `.bash_profile` file or `.bashrc` file. Next, verify the `gpadmin` user can successfully execute commands on all servers without being prompted for a password using the `gpssh` command. For example, the following command executes the `ls -l` command on all servers:  
`gpssh -f hostfile_exkeys -e ls -l $GPHOME.`

## Greenplum Database Installation Overview (Cont)

4

Create the data storage directories as `root` on all servers



- a. Create `/data/master` on the master and standby servers and change ownership of directory to `gpadmin`
- b. Create an exchange key file with segment server host names only
- c. Create `/data/primary` and `/data/mirror` on all segment servers using the exchange key file and change ownership of directories to `gpadmin`

5

Synchronize system clocks across all servers



- a. Configure NTP on the master server so that it points to the data center's NTP time server.
- b. Configure NTP on the standby server so that it points first to the master server and then to the NTP time server.
- c. Configure NTP on the segment servers so that they initially contact the master server if available. If not, they should synchronize with the standby server.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

24

4. As `root`, create the data storage areas on all instances, starting with the master and standby server. These data directories must be owned by the `gpadmin` user. This can be performed with the following commands:

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

Next, create the data storage directories on the segment instances. To do this, create an exchange key file that contains the hostname of each segment instance, one per line. Then, execute the `gpssh` command against that file and create the primary data directory and the mirror data directory on each segment:

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

5. Synchronize system clocks across all servers by configuring network time protocol (NTP) services on the master, standby, and segment instances. This requires setting the service first on the master server to point to the data center's NTP time server. Then configure NTP on the standby server. The standby server should point to the master server first. If the master service is not available, it should point to the data center's NTP time server. All segment servers should first point to the master server and then to the standby server, should the master server become unavailable.

Once the database has been installed, you are ready to test the hardware and operating system values before initializing the database.

## Hardware Verification and Testing

Test the limits of the environment by:

- Establishing baseline disk I/O, CPU performance, and network transfer rates
- Stress testing hardware

Perform the following tests on system components:

- `gpcheckperf`:
  - Test disk input and output rates
  - Test memory bandwidth
  - Test network transfer rates
- `gpcheck`: Validate the OS settings
- `bonnie++`: Stress test for the file system (download from the [bonnie++ website](#))

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

25

When the Greenplum Database is processing large amounts of data and complex DW queries, it will stress hardware to its limits and utilize all of the resources available. It is therefore important to:

- Perform tests to determine the baseline disk I/O, CPU performance, and network transfer rates. It will help you differentiate hardware performance problems from software tuning performance problems.
- Perform stress tests to burn-in your hardware and flush out any bad disks, processors, memory cards, or network interfaces prior to installing Greenplum DB. The first large table load into Greenplum DB will find these problems, so it is better to stress the hardware and uncover these issues first. Even brand new disks can be corrupted.

To validate the configuration and performance of your system, run the following commands:

- `gpcheckperf` – The `gpcheckperf` command performs tests on the:
  - Disk I/O rates using the `dd` command
  - Network performance using the `netperf` command
  - Memory bandwidth using the `stream` test
- `gpcheck` – This command validates the operating system settings for all hosts in the array. The output of the command explains the fixes that must be made to the appropriate host in the array. The command accepts a file with the name of each host in the array per line.
- `Bonnie++` is a free [file system benchmarking tool](#) for [Unix-like operating systems](#), developed by Russell Coker. `Bonnie++` is a benchmark suite that is aimed at performing a number of simple tests of hard drive and file system performance.

## Lab: Systems Preparation and Verification

In this lab, you verify and prepare the operating system for installation and configuration of the Greenplum Database.

You will:

- Install the Greenplum Database binaries
- Prepare data directory locations
- Perform system verification tests

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

26

In this lab, you will prepare your system for installation and then install the Greenplum binaries. You will also verify the state of the system by running verification tests.

## Module 2: Database Installation and Initialization

### Lesson 1: Summary

During this lesson the following topics were covered:

- Greenplum software and hardware solutions and requirements
- Reference architecture for Greenplum
- Verification steps and tools to prepare a system for Greenplum

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

27

This lesson covered the hardware and software solutions available for the Greenplum Database as well as the requirements for the system. The lesson discussed the reference architecture that can be used to build a production-ready environment for the Greenplum Database. The verification steps and tools that are used to prepare a system for the installation, initialization, and day-to-day operations of the Greenplum Database environment were also discussed.

## Module 2: Database Installation and Initialization

### Lesson 2: Greenplum Database Initialization

In this lesson, you initialize the Greenplum environment and examine several array configurations used for Greenplum.

Upon completion of this lesson, you should be able to:

- Initialize the Greenplum Database system
- Identify Greenplum array configurations
- Create mirrors for high availability and redundancy

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

28

In this lesson, you will:

- Continue configuring the Greenplum environment by initializing the database instances.
- Examine the array configurations possible for the Greenplum environment.
- Create mirrors to support high availability and redundancy in the Greenplum environment.

## Greenplum Database System Initialization

To initialize the Greenplum Database:

1. Create a host list file with all segment host names
2. Create the system configuration file, in this example, `gp_init_config`
3. Set the correct locale for the database on the master server
4. Run `gpinitsystem` on the master host

**Example:** `gpinitsystem -c gp_init_config`

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

29

In a Greenplum DB, each database instance, the master and all segments must be initialized across all of the hosts in the system in such a way that they can all work together as a unified DBMS. Greenplum has its own version of `initdb` called `gpinitsystem`, which initializes the database on the master and on each segment instance. This differs slightly from its predecessor, PostgreSQL.

To initialize the Greenplum Database, you must:

- Create a host list file that contains the name of the segment hosts involved in the array.
- Create a configuration file that describes the system setup, including how many segments per host, data directory locations, if mirroring is enabled, and so on. If you are using multiple network interfaces and have multiple segment hostnames per host, it is important that the host file referenced in the configuration has ALL per-interface hostnames for a segment.
- On the master server, define the locale you wish to use for the database. Some locale options cannot be changed after initialization or upgrade, so it is best to verify the setting before.
- Execute the Greenplum Database initialization utility to create the database instance specified in the system configuration file.

After the Greenplum database system has been initialized, it is then ready for use. You can then create and manage databases as you would in a regular PostgreSQL DBMS.

## Greenplum Database Configuration File

```
ARRAY_NAME="Greenplum"
MACHINE_LIST_FILE=/home/gpadmin/gpconfigs/hostfile_gpinitSystem
SEG_PREFIX=gpseg
PORT_BASE=50000
declare -a DATA_DIRECTORY=(/data1/primary /data1/primary
/data1/primary /data2/primary /data2/primary /data2/primary)
MASTER_HOSTNAME=mdw
MASTER_DIRECTORY=/data/master
MASTER_PORT=5432
TRUSTED_SHELL=ssh
CHECK_POINT_SEGMENT=8
ENCODING=UNICODE
#Option Entries for segment mirrors
MIRROR_PORT_BASE=50000
REPLICATION_PORT_BASE=41000
MIRROR_REPLICATION_PORT_BASE=51000
declare -a MIRROR_DATA_DIRECTORY=(/data1/mirror
/data1/mirror /data2/mirror /data2/mirror
/data2/mirror)
```

sdw1-1  
sdw1-2  
sdw1-3  
sdw1-4  
sdw2-1  
sdw2-2  
sdw2-3  
sdw2-4

Pivotal.

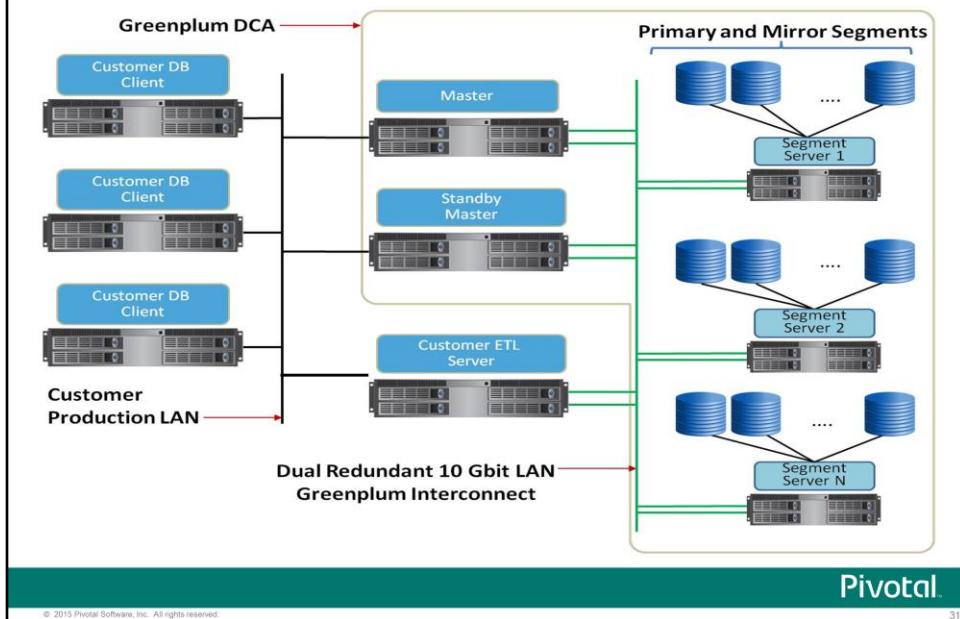
© 2015 Pivotal Software, Inc. All rights reserved.

30

The Greenplum Database Configuration file contains the required parameters necessary for initializing your Greenplum Database. The file defines the following parameters:

- **ARRAY\_NAME** – The unique name of your array
- **MACHINE\_LIST** – File name containing the list of segment hosts in the array
- **SEG\_PREFIX** – The prefix used for the data directories on the master and segment instances
- **PORT\_BASE** – The base port number for the primary segments – this value is incremented by 1 for each primary segment defined
- **DATA\_DIRECTORY** – The data directories for the primary segments
- **MASTER\_HOSTNAME** – The hostname of the master host
- **MASTER\_PORT** – The port number of the master instance
- **TRUSTED\_SHELL** – The type of shell gpinitSystem uses to execute commands on remote hosts
- **CHECK\_POINT\_SEGMENT** – The maximum distance between automatic write ahead logs(WAL) checkpoints
- **ENCODING** – The character encoding to use
- **DATABASE\_NAME** – This optional parameter specifies the database to create on initialization
- **MIRROR\_PORT\_BASE** – The base port number for the mirror segments – this value is incremented by 1 for each mirror segment defined
- **REPLICATION\_PORT\_BASE** – The base number by which the port numbers for the primary file replication process are calculated - this value is incremented by 1 for each primary segment defined
- **MIRROR\_REPLICATION\_PORT\_BASE** – The base number by which the port numbers for the mirror file replication process are calculated - this value is incremented by 1 for each mirror segment defined
- **MIRROR\_DATA\_DIRECTORY** – The data directories for the mirror segments

## Production Greenplum Array



In this configuration, there is one master, one standby master, and two or more segments per host, with one primary segment per CPU core. The network configuration shows that there are multiple network interfaces per segment. Each interface is connected to the interconnect.

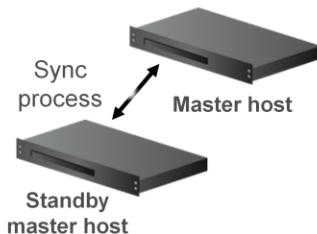
An ETL server is connected directly to the interconnect as well to improve performance for data loads.

This represents one type of array configuration that is commonly used in production environments.

## Creating Mirrors for High Availability and Redundancy

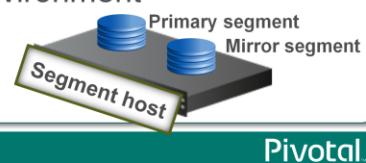
### Master mirroring:

- Lets you create a warm standby master
- Starts the synchronization process between hosts



### Segment mirroring:

- Creates a mirror segment for a primary segment
- Requires enough nodes to spread mirroring
- Can be configured on same array of hosts or hosts in a different environment



Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

32

You can enable mirroring in the Greenplum database during the database initialization phase at setup time, or to an existing system that was configured without mirroring.

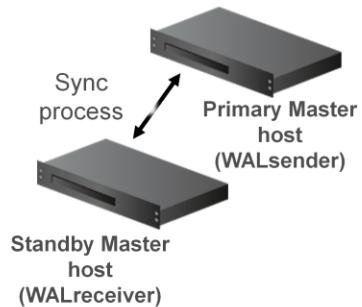
There are two types of mirrors that can be configured:

- **Master mirroring** – When mirroring the master, a warm standby master is created. Once configured, the synchronization or replication process between the master and the standby master is executed. This synchronization process, `gpsyncagent`, is executed from the standby master and ensures that the data between both systems are synchronized. Should the master host become unavailable:
  - The replication process is stopped.
  - The replication logs are used to reconstruct the state of the master at the time of the failure.
  - The standby master can be activated to pick up from the last set of successful transactions completed by the master.
- **Segment mirroring** – A mirror segment is normally configured on a different host than its primary counterpart. It can be configured on systems outside of the array. Changes to the primary segment are copied over to the mirror segment using a file block replication process. Until a failure occurs, there is no live segment instance running on the mirror host, only the replication process. Should the primary segment become unavailable:
  - The file replication process is stopped.
  - The mirror is automatically onlined as a primary segment.

## Warm Standby Master

A warm standby master is:

- A replica of the Greenplum master instance (system catalogs)
- Used to remove single point of failure
- Kept up to date by the *WALsender* (Primary Master) and *WALreceiver* (Standby Master) replication processes
- System Catalogs and Transaction Logs are replicated.



Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

33

A backup or mirror master host serves as a warm standby in the event of the primary master host becoming inoperative. The warm standby master is kept up to date by a transaction log replication process, which runs on the backup master host and keeps the data between the primary and backup master hosts synchronized. The master does not have user data, only system catalog tables. By creating the warm standby master, you ensure that the master is no longer a single point of failure.

You can enable a standby master on a segment host or on a new host. If you are creating the standby master on a host outside of the existing array, verify that the `pg_hba.conf` file of the current master and segments allow connections from the new standby master host. If you add the standby at initialization time, this process is automatically completed for you.

## Adding a Standby Master

### Add a standby master to an existing Greenplum system:

1. Verify Greenplum binaries were installed
2. Source  
`/usr/local/greenplum-db/greenplum_path.sh`
3. Exchange keys
4. Create data directories
5. Initialize the database with the following:  
`gpinitstandby -s standby_hostname`

### Add a standby master during initialization:

1. Verify Greenplum binaries were installed
2. Source  
`/usr/local/greenplum-db/greenplum_path.sh`
3. Create data directories
4. Initialize Greenplum and specify the warm standby master with the following:  
`gpinitsystem -c gp_init_config -s standby_hostname`

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

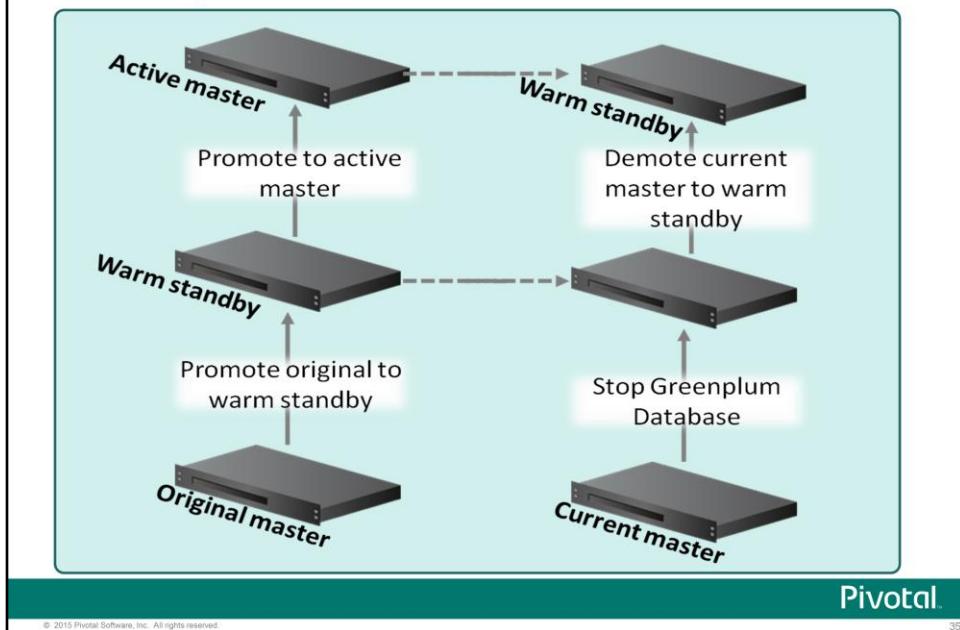
34

To add a standby master to an existing Greenplum database system:

- Verify Greenplum has already been installed and configured on the standby master and that host keys have already been exchanged. This includes ensuring that the superuser account has been created, the binaries for Greenplum installed, the variables configured, keys exchanged, and data directories created on the standby master.
- Run `gpinitstandby` on the active primary master. Include the `-s` option followed by the host name of the standby master.

At initialization time, include the `-s` option followed by the hostname of the standby master when executing the `gpinitsystem` command.

## Primary Master Failure and Restoration



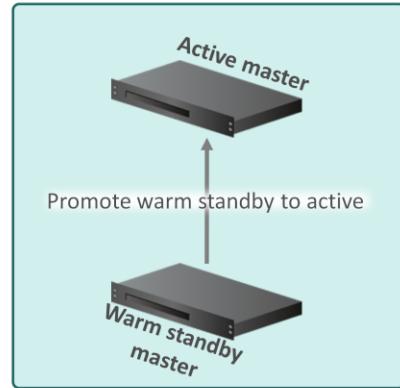
The warm standby master offers a measure of protection for your database by assuming the role of primary should the original primary master fail. The synchronization process, `WALsender` and `WALreceiver`, allows the standby to be engaged with little to no loss of data by keeping the transaction log data between the primary and standby masters synchronized. In-flight transactions may need to be restarted.

When the primary master fails, the warm standby is manually promoted as the active master. Once the primary master becomes available, it can be promoted to its original role, demoting the warm master from the active mode to its original warm standby role.

## Promoting a Warm Standby to an Active Primary Master

### Promote a standby master to primary:

1. On the standby master, run:  
`gpactivatestandby -d  
$MASTER_DATA_DIRECTORY`
2. Optionally, configure a new standby that has already been configured:  
`gpactivatestandby -d  
$MASTER_DATA_DIRECTORY  
-c new_standby_hostname`
3. Verify the active master is Active and the standby is Passive, if configured:  
`gpstate -f`



Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

36

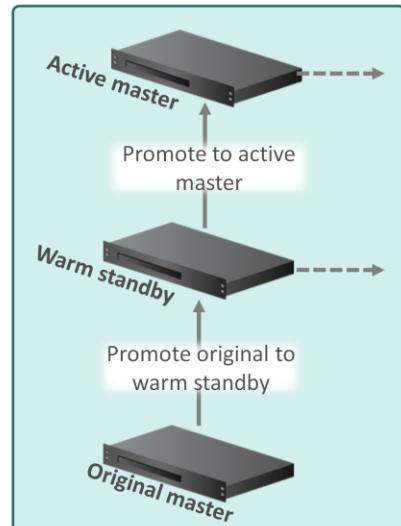
To promote the warm standby you configured to be the master:

1. Run the `gpactivatestandby -d $MASTER_DATA_DIRECTORY` command on the warm standby. This promotes the warm standby to now be the active primary master.
2. Optionally, define a new warm standby with the `gpactivatestandby` command. The `-c` option allows you to specify the hostname to the new warm standby.
3. Once you have promoted the warm standby to be an active primary master, verify the state of the servers with the `gpstate -f` command.

## Promoting the Original Master to the Active Master

### Promote the original failed master back to a master:

1. Fix problems on the original standby and verify the Greenplum Database processes have not started.
2. Initialize the original master as a standby master:  
`gpinitstandby -s  
original_master_hostname`
3. Stop the Greenplum Database on the current master:  
`gpstop -m`



Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

37

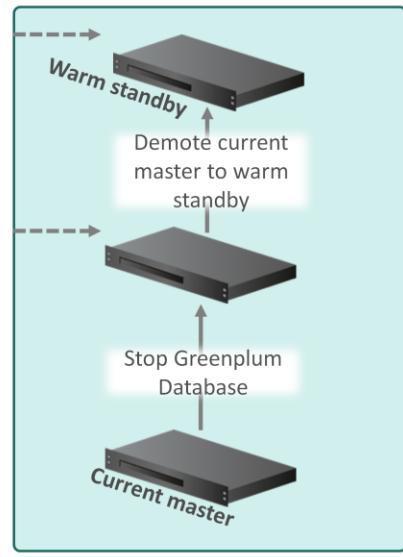
If the original master has been restored, you can promote it back to an active master with the following steps:

1. Ensure that the Greenplum Database is not running on the original master. The instance should already be running on the server currently acting as the primary master.
2. Initialize the original master so that it is now a warm standby. This is done with the `gpinitstandby` command with the `-s` option that allows you to specify the name of the original master.
3. Stop the Greenplum database on the current master using the `gpstop -m` command. The `-m` option stops only the master.

## Promoting the Original Master to the Active Master (Cont)

### Promote the original failed master back to a master:

4. Promote the original master from a standby to the master:  
`gpactivatestandby -d  
$MASTER_DATA_DIRECTORY`
5. Reinitialize the original standby master to be the standby master again:  
`gpinitstandby -s  
original_standby_master_hostname`
6. Check the state of the master and standby master:  
`gpstate -f`



Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

38

4. Promote the original master, which is now acting as a warm standby, to be the active primary. Use the `gpactivatestandby` command to promote the original master.
5. Initialize the master that was acting as a primary master to now act as a standby master. Use the `gpinitstandby` command to demote this server.
6. Check the state of the master and current standby servers using the `gpstate -f` command. The original master should now show as Active and the standby status should now be Passive.

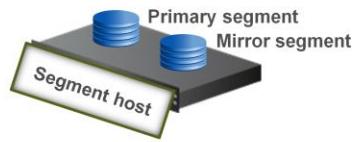
## Mirror Segments

Mirror segments are:

- A replica of a given primary segment
- Used for data redundancy

Mirroring can be enabled:

- At array initialization time
  - (set parameters in `gp_init_config` file)
- On an active Greenplum Database system:
  - `gpaddmirrors` command



Mirror segments can be deployed:

- On the same hosts as your primary segments (Not recommended)
- On a different set of host than your primary segments (Spread Mirroring)

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

39

Mirrored segments replicate a primary segment counterpart providing redundancy and fault tolerance support for your data. Database queries can fail over to a mirror segment should the primary segment be unavailable.

Mirroring can be enabled:

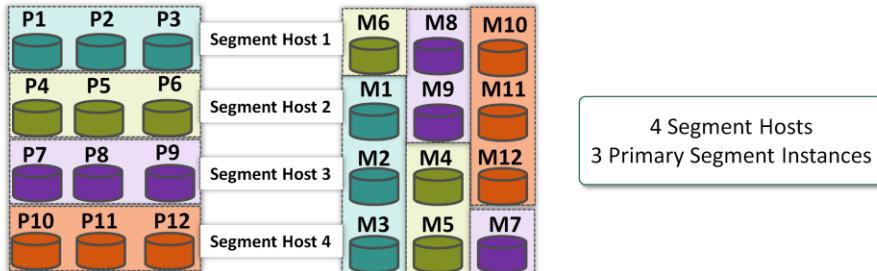
- At array initialization by setting parameters in the `gp_init_config` system configuration file you defined earlier.
- On an active Greenplum Database using the same hosts as the primary segments by:
  1. Creating data storage for the mirror on a different file system than the primary segment is using
  2. Exchanging SSH keys as segment hosts must be able to SSH and SCP to each other without a password
  3. Running the `gpaddmirrors` utility with the `-p` option, followed by the number to add to your primary port segment number to determine the port number for the mirror

## Mirror Segments (continued)

- On an active Greenplum Database using different hosts than the primary segments by:
  1. Verifying all segments have Greenplum installed
  2. Allocating data storage for mirror segments on all segment hosts
  3. Exchanging SSH keys
  4. Creating a configuration file with the host names, ports, and data directories where you want your mirrors created. You can use `gpaddmirrors -o filename` to create a sample file that you can edit with your true parameters.
  5. Running the `gpaddmirrors -i filename`, where `filename` is the name of the file that contains the mirror configuration.

**Note:** If using a multi-NIC configuration (multiple host names per segment host), DO NOT enable mirrors at initialization time. Do so after you have initialized and edited the system catalog. You can then run `gpaddmirrors`.

## Mirroring in Greenplum – Spread Mirror Distribution



- Spreads the mirror segments across the available hosts
- Mirror spreading will place each mirror on a different host within the Greenplum Database array
- Spreading is only allowed if there is a sufficient number of hosts in the array
  - Number of hosts is greater than the number of segment instances

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

41

In this example, we focus on the spread mirror distribution for 4 segment hosts with 3 primary segments per host.

In spread mirror distribution, mirror segments are spread across the segment servers so that no single server contains more than one mirror for another segment server. This spreads the mirror segments across the available hosts, balancing the mirrors and reducing the chances that the failure of a single segment will impact the database.

Mirror spreading will place each mirror on a different host within the Greenplum database array. Spreading is only allowed if there is a sufficient number of hosts in the array. To meet this requirement, the number of hosts must be greater than the number of segment instances.

Note that cluster expansions and upgrades need to be carefully planned out to insure proper spreading of new mirrors across all the servers in the cluster.

## Fault Detection and Recovery

Fault detection:

- Is handled by `ftsprobe`
- Marks a segment as down when a connection fails or a response timeout is exceeded.
- Allows subsequent connection requests to switch to the mirror and succeed
- Requires Greenplum administrators to manually recover a segment marked invalid with `gprecoverseg`
- May also require that the Greenplum administrator has to manually rebalance the database cluster with a `gprecoverseg -r`
- Requires vigilance from Greenplum administrators

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

42

Fault detection is handled by `ftsprobe`, a Greenplum Database server subprocess. This fault detection process monitors the Greenplum array, scanning all segments and database processes at configurable intervals.

Whenever the fault detection process cannot connect to a segment, it marks that segment instance as down in the Greenplum Database system catalog. Once a segment is down, it will remain out of operation until an administrator initiates the recovery process to bring that segment back online.

When mirroring is enabled in a Greenplum Database system, the system will automatically failover to the mirror copy whenever a primary copy becomes unavailable. A Greenplum Database system can remain operational if a segment instance or host goes down as long as all portions of data are available on the remaining active segments.

## Fault Detection and Recovery (continued)

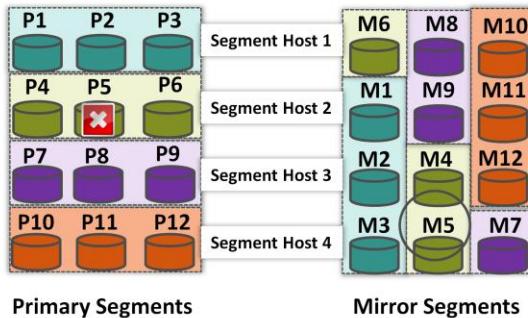
To recover failed segments in the system, the Greenplum administrator uses the `gprecoverseg` recovery utility. This utility:

- Locates the downed segments
- Checks if they are valid
- Compares the transactional state with the currently active segment to find out what changes were missed when the segment was offline.
- Synchronizes only the changed database files with the active segment
- Brings the segment back online.

This recovery process is performed while the Greenplum Database system is up and running.

If you do not have mirroring enabled, the system will automatically shut down if a segment instance becomes invalid. You must manually recover all failed segments before operations can continue.

## Mirroring in Greenplum – Failed Primary Segment Example



Primary Segment Failure:

1. The `ftsprobe` process on the master detects the segment down and marks it invalid.

Pivotal

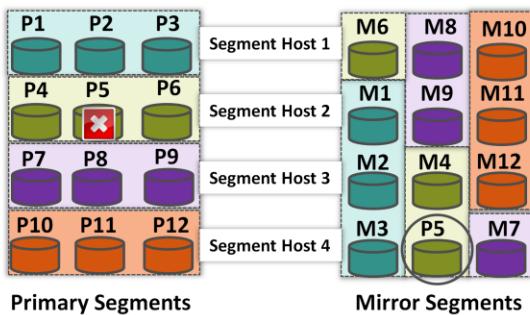
© 2015 Pivotal Software, Inc. All rights reserved.

44

Over the next few slides, we will examine how the failure of a segment is detected and handled by Greenplum Database.

Primary segment P5 on segment host 2 is marked down and invalid. The `ftsprobe` process initiates a failover to mirror segment M5 mirror on segment host 4.

## Mirroring in Greenplum – Failed Primary Segment Example



### Primary Segment Failure:

1. The `ftsprobe` process on the master detects the segment down and marks it invalid.
2. The mirror segment is validated to ensure that it was synchronized with its primary segment. Mirror segment becomes the primary.

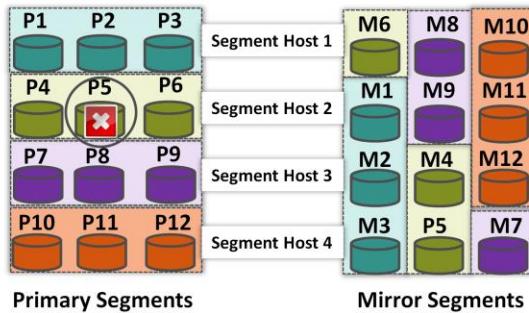
Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

45

Mirror segment M5, on segment host 4, is validated to be in sync with the corresponding primary that became unavailable. The M5 mirror is now promoted to become the primary segment P5.

## Mirroring in Greenplum – Failed Primary Segment Example



Primary Segment Failure:

- Once it has been determined why the original primary P5 went down and is repaired, you can then bring that segment back online to become the mirror segment to protect the new primary P5.

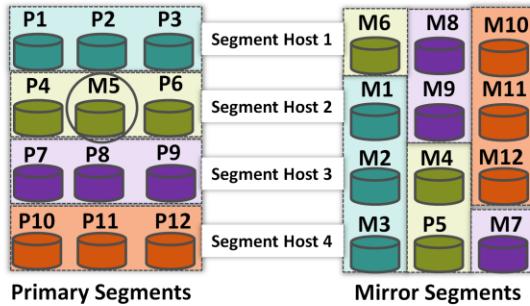
Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

46

Once repaired and made available, the original P5 primary segment can be brought back online. When brought back online it will become the mirror M5, protecting the new P5 primary segment.

## Mirroring in Greenplum – Failed Primary Segment Example



### Primary Segment Failure:

4. Once it has been determined why the original primary P5 went down and is repaired, you can then bring that segment back online to become the mirror segment to protect the new primary P5.
5. Run the `gprecoverseg` command to bring back up the down segment. The old primary P5 becomes the M5 mirror.

Pivotal

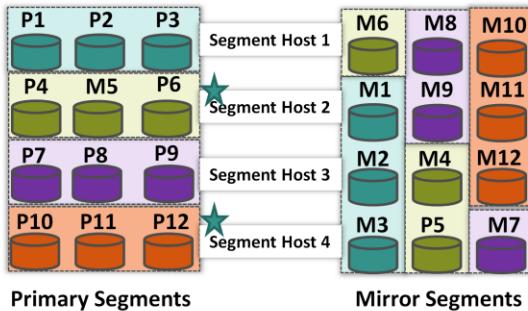
© 2015 Pivotal Software, Inc. All rights reserved.

47

Use the `gprecoverseg` command to restore the original roles of the primary and mirror segments. The database stores information on the roles as originally configured as well as their current status in the `gp_segment_configuration` table. By issuing the `gprecoverseg` command, Greenplum:

- Brings recovered segments back online.
- Restores all segments to their original roles
- Verifies that all segments are valid
- Synchronizes the changed database files with the active segment

## Mirroring in Greenplum – Re-balancing Segments



The database cluster is in an un-balanced condition

- Segment Host 2: Two Primaries, 4 Mirrors
- Segment Host 4: 4 Primaries, 2 Mirrors

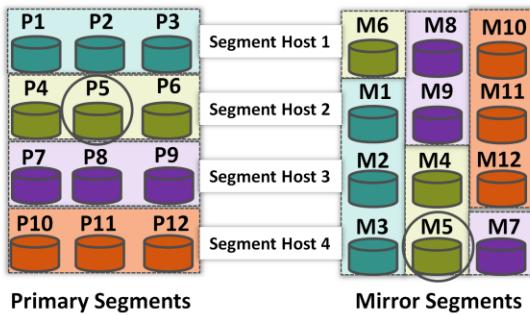
Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

48

Before rebalancing the segments, segment host 2 and segment host 4 are in an unbalanced condition. Segment host 2 has 2 primaries and 4 mirrors, and segment host 4 has 4 primaries and 2 mirrors. From a processing of queries perspective, segment host 4 will be processing more and could cause query performance issues. To re-balance the database cluster, run `gprecoverseg -r` command to accomplish this.

## Mirroring in Greenplum – Re-balancing Segments



The database cluster is in an un-balanced condition

- Segment Host 2: Two Primaries, 4 Mirrors
- Segment Host 4: 4 Primaries, 2 Mirrors

Run the following command to re-balance the cluster

- `gprecoverseg -r`

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

49

Once `gprecoverseg` has successfully executed, the original roles are assigned to the segments.

## Setting Greenplum Environment Variables

The terminal window shows the contents of the .bash\_profile file for the gpadmin user. The file includes aliases, functions, and environment variable definitions. Annotations explain the purpose of each variable:

- GPHOME** points to the base Greenplum directory (executables and libraries)
- PGDATABASE** sets up your default database to connect to
- Always source the *greenplum\_path.sh* file (sets up paths to executables and libraries)
- MASTER\_DATA\_DIRECTORY** is the location of the data directory on the master server

```
gpadmin@mdw:~$ .bash_profile
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs
PATH=$PATH:$HOME/bin
export PATH

GPHOME=/usr/local/greenplum-db
export GPHOME
MASTER_DATA_DIRECTORY=/data/master/gpseg-1
export MASTER_DATA_DIRECTORY
PGDATABASE=gpadmin
export PGDATABASE
source $GPHOME/greenplum_path.sh
~/.bash_profile" 20L, 364C written
1,1
```

Pivotal  
© 2015 Pivotal Software, Inc. All rights reserved.

Once you have initialized the Greenplum Database, you must configure the Greenplum environment by setting up or accessing several variables that will be used when you access or administer the database.

To configure the environment, perform the following on the master and standby servers:

- Set the value of the **MASTER\_DATA\_DIRECTORY** variable to the location of the data directory on the master server. This variable is used when starting, stopping, or otherwise accessing the Greenplum Database.
- Source the contents of the /usr/local/greenplum-db/greenplum\_path.sh file. This will set the value of other environment variables used by Greenplum.

Add these settings to the **.bash\_profile** or **.bashrc** file for the **gpadmin** user. By adding it to the **.bash\_profile** file, the variables will be read once when you log into the system as the **gpadmin** user. If you add it to the **.bashrc** file, it will be read for each separate shell session you initiate as the **gpadmin** user.

Optionally, you can configure several other variables that will override the behavior of the Greenplum client, **psql**. These variables will be explained in greater detail later in the course.

## Lab: Pivotal Greenplum Database Initialization

In this lab, you will perform the following installation and setup tasks necessary for the Greenplum Database software to run:

- Initialize Greenplum Database without mirrors and the standby server
- Delete the configured environment and initialize Greenplum Database with mirrors and the standby server

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

51

In this lab, you will continue configuring your Greenplum database environment by initializing the Greenplum Database system. You will also configure mirrors for the primary segments you have configured as part of the lab.

## Module 2: Database Installation and Initialization

### Lesson 2: Summary

During this lesson the following topics were covered:

- Initialize the Greenplum Database system
- Identify Greenplum array configurations
- Create mirrors for high availability and redundancy

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

52

This lesson covered how to initialize the Greenplum Database environment, by configuring all components at once, or by configuring just the master and primary segments for the environment. Different array configurations were shown for production and standalone or testing environments. Finally, a discussion on creating and recovering mirrors for high availability and redundancy were also discussed.

## Module 2: Summary

Key points covered in this module:

- Verification of system performance using available verification tools
- Initialization and validation of a Greenplum installation

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

53

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

- Verify system performance by using available verification tools after installing the Greenplum binaries.
- Initialize and validate the Greenplum installation.

This slide is intentionally left blank.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

54

# Module 3: Greenplum Database Tools, Utilities, and Internals

This module presents reference architecture information and shows you how to install and configure your Greenplum Database.

Upon completion of this module, you should be able to:

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

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

1

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

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

## Module 3: Greenplum Database Tools, Utilities, and Internals

### Lesson 1: Using the PSQL Client and Greenplum Utilities

In this lesson, you use PSQL to connect to and issue SQL commands. You learn the meta commands available with PSQL and the Greenplum utilities you can use to administer your Greenplum Database.

Upon completion of this lesson, you should be able to:

- Connect to the Greenplum Database using PSQL
- Issue SQL commands from PSQL
- Issue PSQL meta commands from PSQL
- Identify Greenplum utilities to maintain the database

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

2

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

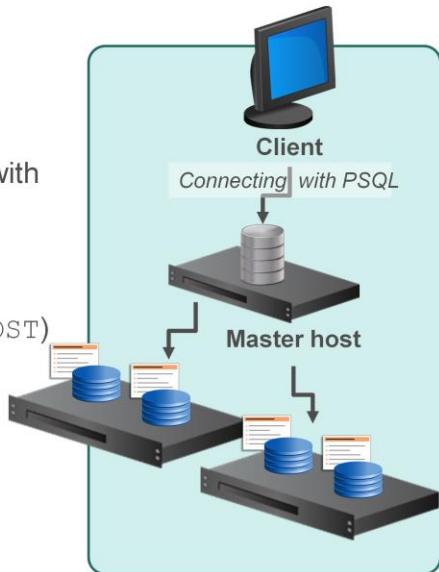
In this lesson, you:

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

## Connecting with PSQL

Using PSQL, you:

- Connect through the master
- Specify connection information with the following options:
  - database name (`-d | PGDATABASE`)
  - master host name (`-h | PGHOST`)
  - master port (`-p | PGPORT`)
  - user name (`-U | PGUSER`)
- Connect the first time to:
  - template1 database
  - default superuser account (`gpadmin`)



Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

3

PSQL:

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

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

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

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

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

## Issuing SQL Commands

Commands can be issued in:

- **Interactive mode:**

```
psql mydatabase  
mydatabase=# SELECT * FROM foo;
```

- Non-interactive mode lets you run a semicolon separated list of commands

```
psql mydatabase -ac "SELECT * FROM foo;"
```

- Non-interactive mode lets you run multiple commands

```
psql mydatabase -af  
/home/lab1/sql/createdb.sql
```

Use semi-colon (;) to denote end of a statement

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

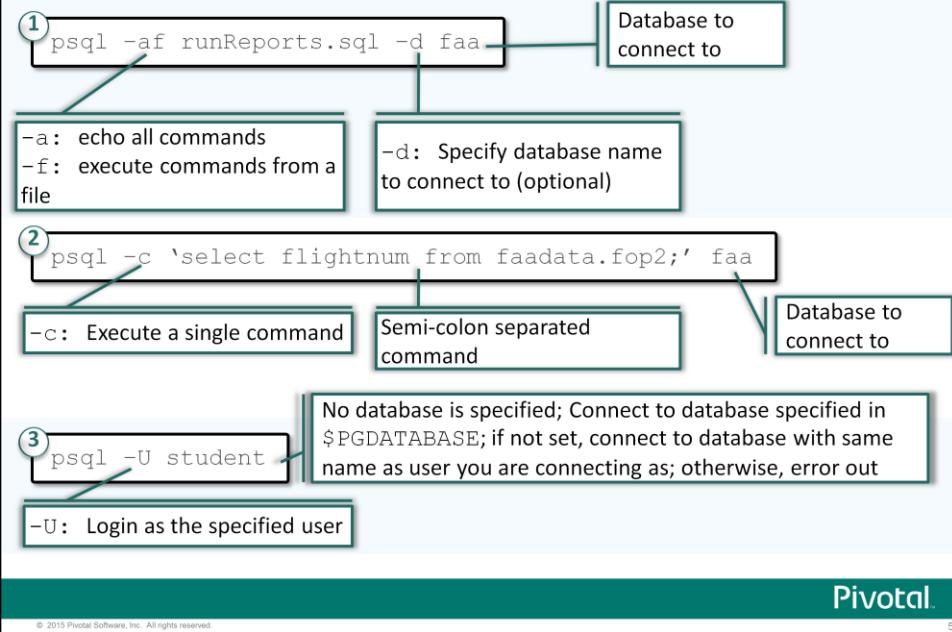
4

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

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

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

## PSQL Examples



Here are some examples of the PSQL command.

The first example is run in non-interactive mode. All of the commands are stored in the SQL file, runReports.sql. The `-a` option echoes all commands provided, while the `-f` option allows you to specify a file name that contains the commands and exit once complete. The

`-d` option allows you to specify the database you are connecting to when executing the command. This is followed by the database name. You do not have to use the `-d` option when specifying a database name.

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

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

## Common PSQL Meta-Commands

Commonly used PSQL meta-commands include:

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

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

6

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

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

## Greenplum Client Utility Applications

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

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

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

7

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

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

## Greenplum Database Utilities

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

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

8

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

## Starting and Stopping the Database

gpstart Options		gpstop Options	
Shared Options		Description	
-R		Restricted mode	
-v		Verbose mode	
-?		Obtain help for the utility	
		-M fast	
		Fast shutdown where in-flight transactions are rolled back	
		-M smart	
		Shut down the database if there are no active connections (default)	
		-r	
		Restart the database	
		-u	
		Reload pg_hba.conf and postgresql.conf	

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

9

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

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

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

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

## Obtaining Database Status

```
[gpadmin@mdw: ~]$ gpstate
[gpadmin@mdw: ~]$ gpstate -b
[gpadmin-[INFO] : obtaining update with -b...
[gpadmin-[INFO] : local Greenplum Version: 'postgres (Greenplum Database) 4.3.4.0 build 1'
[gpadmin-[INFO] : master Greenplum Version: 'PostgreSQL 8.2.15 (Greenplum Database 4.3.4.0 build 1) on x86_64-unk
[gpadmin-[INFO] : -c 4.4.2
[gpadmin-[INFO] : -obtaining Segment details from master...
[gpadmin-[INFO] : -gathering data from segments
[gpadmin-[INFO] : -Greenplum instance status summary
[gpadmin-[INFO] : -Master instance
[gpadmin-[INFO] :   Master standby = Active
[gpadmin-[INFO] :   Standby master state = Smdw
[gpadmin-[INFO] :   Total segment instance count from metadata = 4
[gpadmin-[INFO] : - Primary Segment Status
[gpadmin-[INFO] :   Total primary segments = 2
[gpadmin-[INFO] :   Total primary segment valid (at master) = 2
[gpadmin-[INFO] :   Total primary segment failures (at master) = 0
[gpadmin-[INFO] :   Total number of postmaster.pid files missing = 0
[gpadmin-[INFO] :   Total number of postmaster.pid PIDs missing = 0
[gpadmin-[INFO] :   Total number of postmaster.pid PID found = 2
[gpadmin-[INFO] :   Total number of /tmp lock files missing = 0
[gpadmin-[INFO] :   Total number of /tmp lock file found = 2
[gpadmin-[INFO] :   Total number postmaster processes missing = 0
[gpadmin-[INFO] :   Total number postmaster processes found = 2
[gpadmin-[INFO] : - Mirror Segment Status
[gpadmin-[INFO] :   Total mirror segments = 2
[gpadmin-[INFO] :   Total mirror segment valid (at master) = 2
[gpadmin-[INFO] :   Total mirror segment failures (at master) = 0
[gpadmin-[INFO] :   Total number of postmaster.pid files missing = 0
[gpadmin-[INFO] :   Total number of postmaster.pid PIDs missing = 0
[gpadmin-[INFO] :   Total number of postmaster.pid PID found = 2
[gpadmin-[INFO] :   Total number of /tmp lock files missing = 0
[gpadmin-[INFO] :   Total number of /tmp lock file found = 2
[gpadmin-[INFO] :   Total number postmaster processes missing = 0
[gpadmin-[INFO] :   Total number postmaster processes found = 2
[gpadmin@mdw: ~]$
```

Pivotal  
© 2015 Pivotal Software, Inc. All rights reserved.

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

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

## gpstate Utility Options

Option	Description
-c	Primary to mirror mappings
-e	Display segments with potential issues with regards to the mirrors
-f	Display standby master details, if configured
-i	Display Greenplum Database version
-m	List the mirrors in the system
-p	Show the Greenplum system ports
-s	Show detailed information on segments
-Q	Quick status check for downed segments
Shared Options	Description
-B #	The number of segments to start or stop in parallel
-d <master_data_directory>	The data directory for the master server
-q	Quite mode with no screen output
-v	Verbose output

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

11

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

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

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

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

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

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

## Lab: Using the PSQL Client and Greenplum Utilities

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

You will:

- Connect to the database using `psql`, the command-line client interface to the Greenplum Database
- Use `psql` to run SQL commands both in interactive mode and non-interactive mode
- View the help in `psql` and about `psql` meta-commands

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

12

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

## Module 3: Greenplum Database Tools, Utilities, and Internals

### Lesson 1: Summary

During this lesson the following topics were covered:

- Connecting to the Greenplum Database using PSQL
- Issuing SQL commands from PSQL
- Issuing PSQL meta commands from PSQL
- Greenplum utilities used to maintain the database

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

13

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

## Module 3: Greenplum Database Tools, Utilities, and Internals

### Lesson 2: Pivotal Greenplum Command Center

In this lesson, you examine the benefits and features of the Pivotal Greenplum Command Center, using it to view resources and queries in the Greenplum environment.

Upon completion of this lesson, you should be able to:

- Describe the purpose of Pivotal Greenplum Command Center and identify its features and benefits
- List the three main components of the Greenplum Command Center architecture
- List the high-level steps required when installing and configuring the Pivotal Greenplum Command Center software

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

14

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

In this lesson, you will:

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

## Greenplum Command Center Overview

Greenplum Command Center:

- Collects system metrics and query details
- Aggregates historical information
- Monitors key metrics
- Controls instance power
- Modifies resource management

The screenshot displays the Greenplum Command Center interface, which includes several key components:

- System metrics:** A dashboard showing real-time performance metrics such as CPU usage, memory, and disk I/O.
- Real time metrics by server:** A section showing detailed real-time metrics for each database server.
- Query plan details:** A table showing the execution details of various database queries.
- Administrative Tasks:** A section for managing database segments, replication status, and preferred roles.

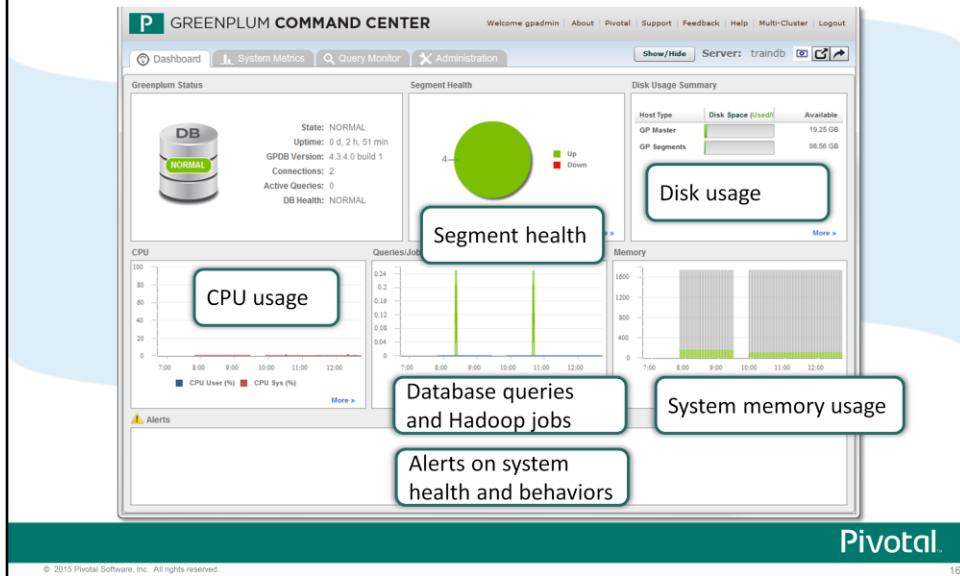
The interface is branded with "Pivotal" at the bottom right.

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

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

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

# Monitoring and Managing with Greenplum Command Center



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

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

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

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

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

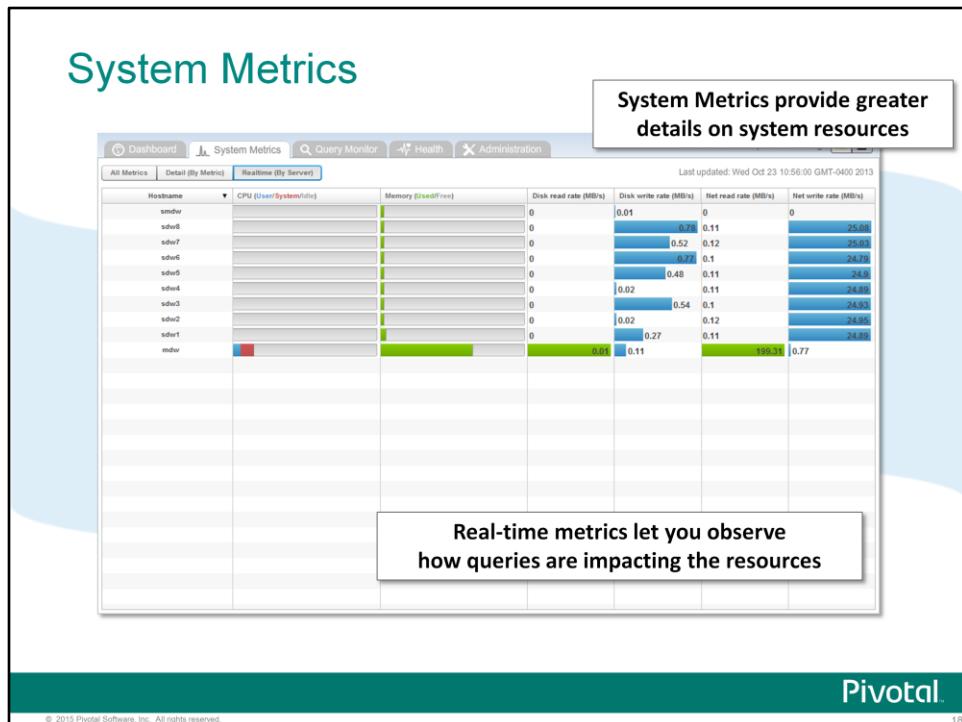
You can also obtain information on active queries and the resources that they are consuming. From this page, you can drill-down into greater detail on each query.

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

## Greenplum Command Center for the Pivotal Data Computing Appliance

The screenshot shows the Greenplum Command Center interface. At the top, there are tabs for Dashboard, System Metrics, Query Monitor, Health, and Administration. The Server is set to "Greenplum Training". On the left, a sidebar under "Administration" includes Database Admin, Segment Health (which is selected), Database Usage Report, Storage Monitoring, Workload Management, and Manage Resource Queues. The main area is titled "Segments (96)". It contains three circular dashboards: "Segment Status" (96 Up, 0 Down), "Replication Status" (96 Synced, 0 Resyncing, 0 Change Tracking, 0 Not Syncing), and "Preferred Role" (06 Preferred, 0 Not Preferred). Below these are three tables: "Segment Status", "Replication Status", and "Content ID". The "Content ID" table lists 96 rows of segment information, including Hostname, Address, Port, Replicab, DBID, Content ID, Status, Role, Preferred, Mode, Recovered, SAN Mou, and Last Event. A callout box highlights the "Segment health, rebalancing, and recovery" section at the bottom left. The footer includes a "Pivotal" logo and copyright information: "© 2015 Pivotal Software, Inc. All rights reserved." and "17".

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



Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

18

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

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

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

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

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

All Active Queries: 5 Running, 0 Queued

Selected	Query ID	Submit Time	Username	Database	Status	Wait time (s)	Run time (s)	Rows out	CPU total	Row skew	Priority	Queue name	Details	Query plan
	1324009736-2434467	Dec-23-2011, 5:16:30 PM	gadmin	tpn182	running	0	30	0	13.8151	313.49	0	MAX	pg_default	
	1324009736-2434447	Dec-23-2011, 5:16:30 PM	gadmin	tpn182	running	0	30	0	18.8472	309.65	0.38	MAX	pg_default	
	1324009736-2434467	Dec-23-2011, 5:16:30 PM	gadmin	tpn182	running	0	30	0	15.2893	371.65	15.89	MAX	pg_default	
	1324009736-2434467	Dec-23-2011, 5:16:30 PM	gadmin	tpn182	running	0	30	0	10.8275	360.52	3.85	MAX	pg_default	
	1324009736-2434477	Dec-23-2011, 5:16:30 PM	gadmin	tpn182	running	0	30	0	11.8386	253.9	17.22	MAX	pg_default	

Search by:

Filter by:

- All Active Queries
- My Active Queries

Start Time:

End Time:

Username:

Database:

Min. Runtime (s):

Query Status:

- Inactive
- Aborted

Search

Query monitoring screen

Block

Node type: Gather Motion

Query plan details

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

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

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

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

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

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

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

# Database Administration and Management

The screenshot shows the 'Database Admin' section of the Greenplum Command Center. At the top, there's a navigation bar with links for Dashboard, System Metrics, Query Monitor, Health, and Administration. The 'Administration' tab is selected. Below the navigation is a sidebar with links for Administration, Database Admin (which is selected), Segment Health, Database Usage Report, Storage Monitoring, Workload Management, and Manage Resource Queues. The main content area is titled 'Database Admin' and shows 'Greenplum Database' with a status of 'UP'. It includes a 'Smart' dropdown menu and buttons for 'Stop', '- OR -', and 'Restart'. A note below the buttons says: 'Select option for stopping or restarting GPOD. [Smart] If there are active connections, this command fails with a warning. This is the default shutdown mode.' A callout box highlights the 'Start and stop the database' functionality. The bottom of the page has a teal footer with the Pivotal logo and copyright information.

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

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

**Database and Storage Usage Reports**

**View database usage report**

**Storage utilization reports**

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved. 21

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

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

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

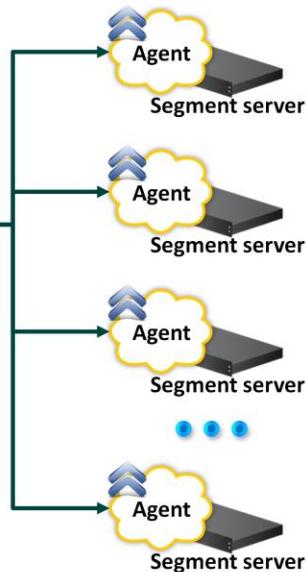
The Workload Management screen allows you to view and make changes to the Greenplum Database configuration parameters for managing workload and resources. The parameters are displayed at both the master and local, or segment, levels and are classified according to their potential effect on the database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

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

## Greenplum Command Center Architectural Overview



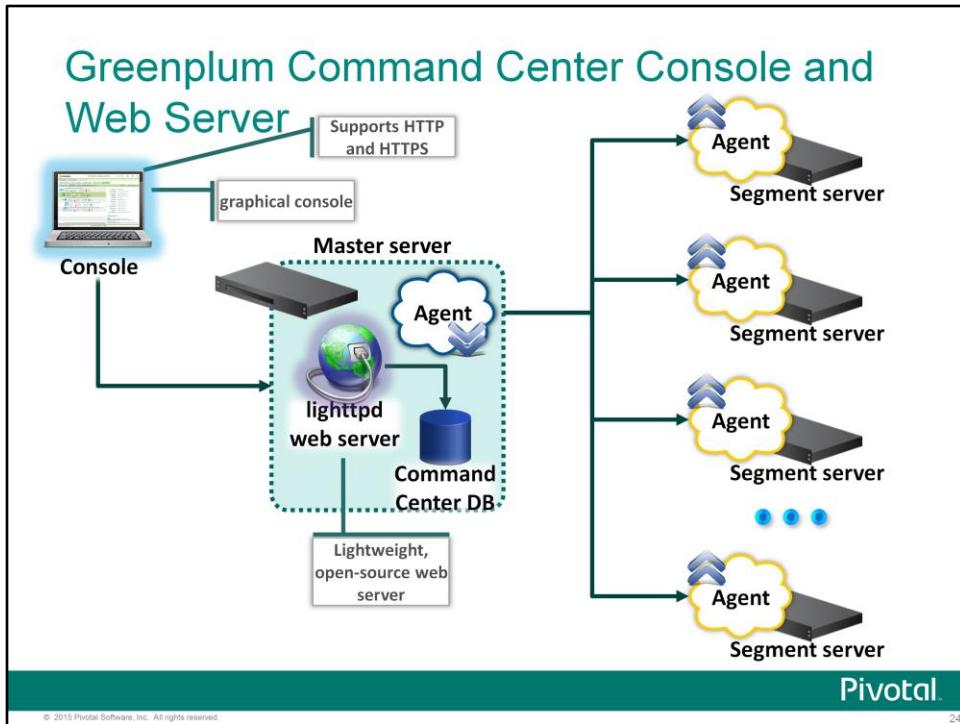
Console



Pivotal.

Greenplum Command Center is comprised of:

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

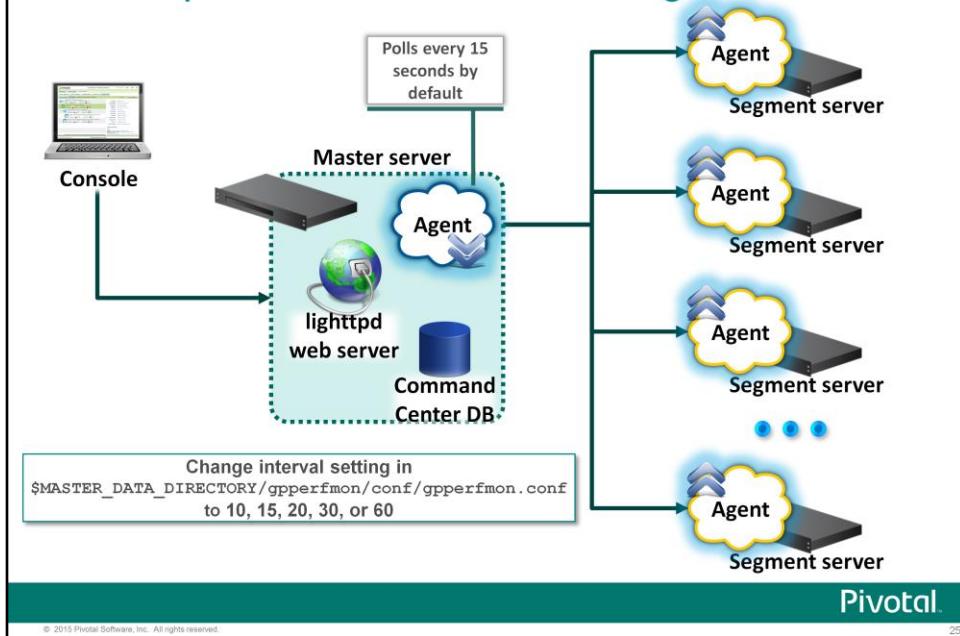


The Greenplum Command Center Console consists of a flash-based graphical console accessed through a browser and used to view performance metrics for all servers in the cluster. You can access separate Greenplum instances from the console by assigning each instance its own unique port ID.

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

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

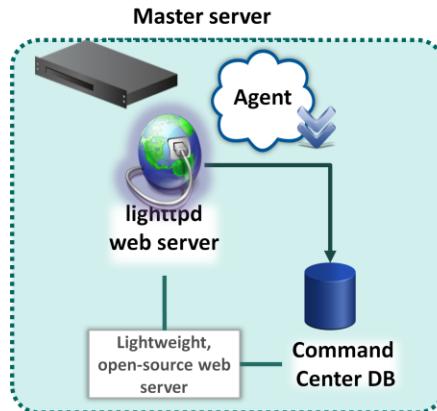
## Greenplum Command Center Agents



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

## Greenplum Command Center Database

Command Center stores the data collected by the agents in to the gpperfmon database under the gpmn role.



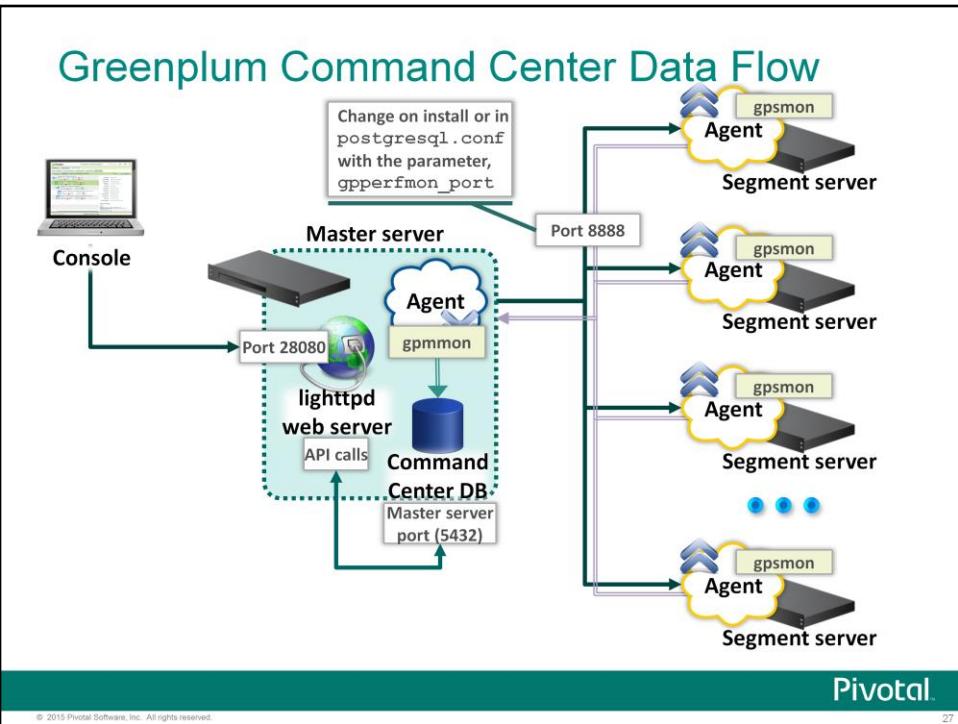
Pivotal

26

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

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

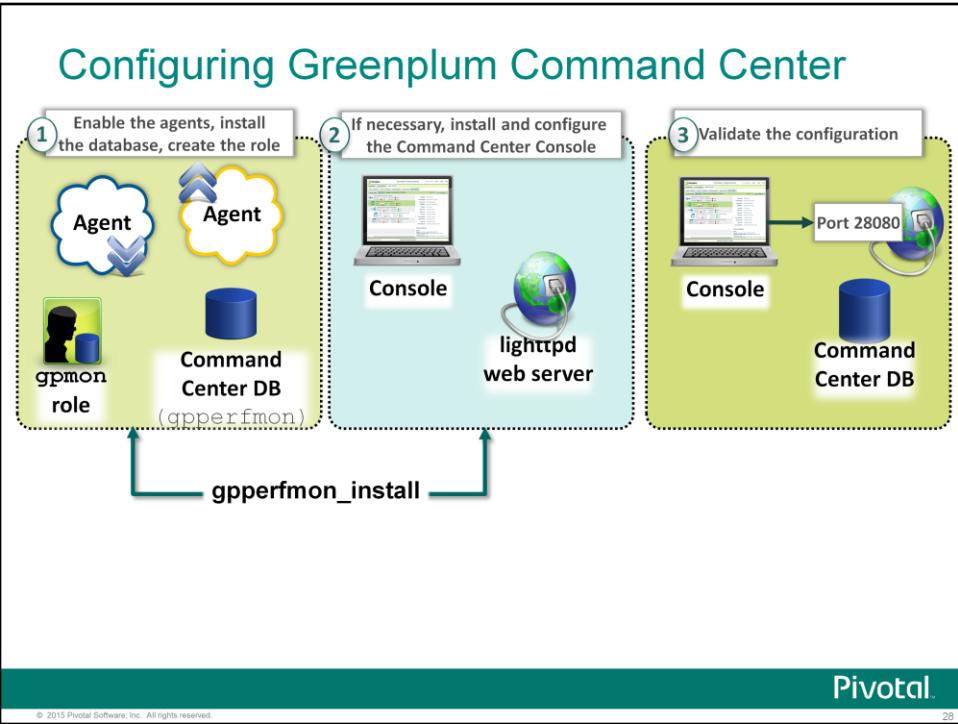
Note that while the gpperfmon database can provide historical data, it should not be used for accounting as it does not trap all events that have happened. What you are seeing is a sampling provided over a period of time. This is to prevent Greenplum Command Center from hogging system resources which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



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

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

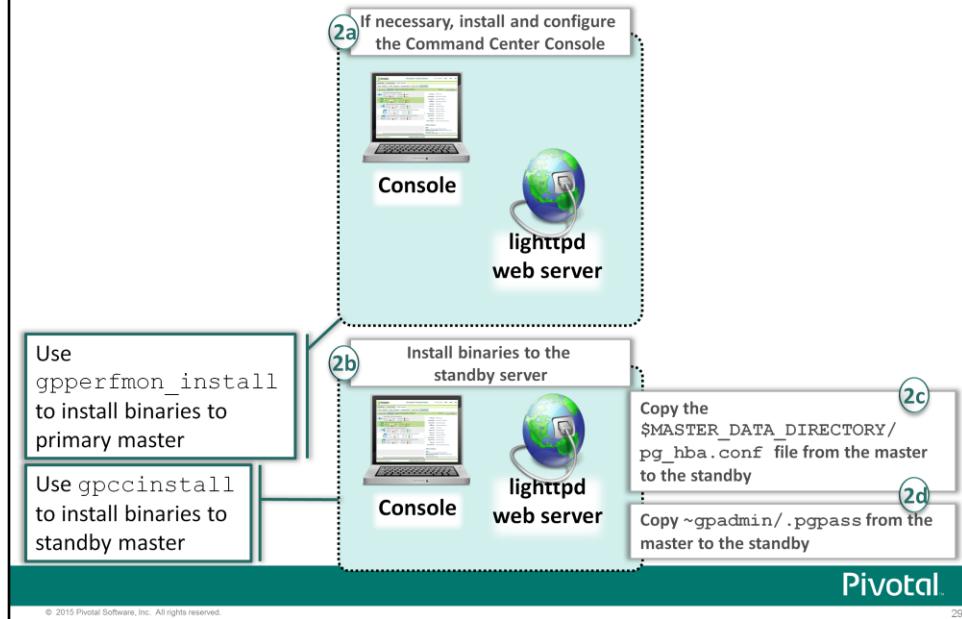
Information is written to `now` files which are available to the external tables. After a period of time, data collected in the `now` files are written to the `tail` files. This information will eventually be committed to the Command Center database, `gpperfmon`.



You perform three main tasks when enabling Greenplum Command Center:

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

## Configuring the Standby Server



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

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

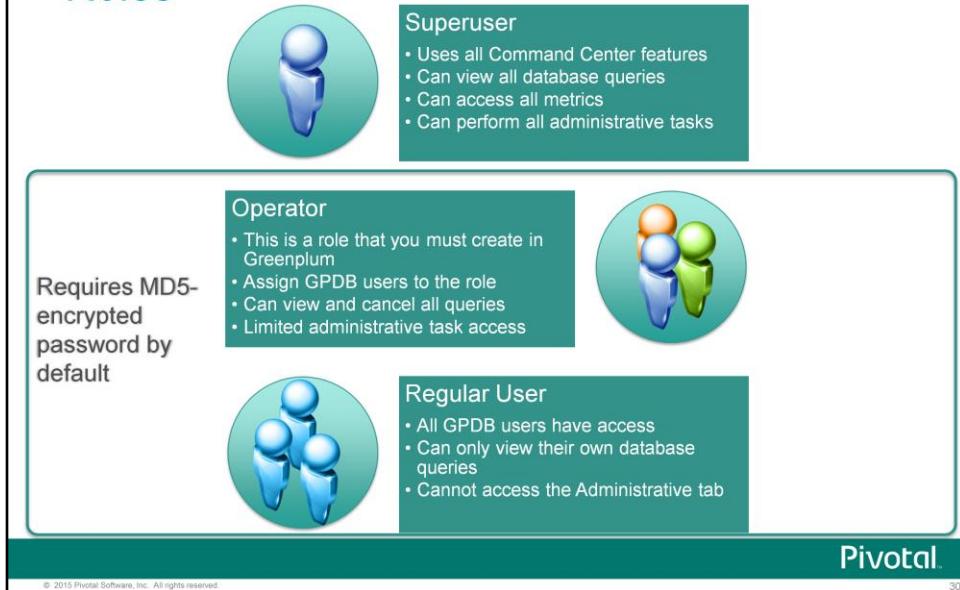
Assuming the filename is `gpcc_hosts`, the command to perform the installation is:

```
$ gpccinstall -f gpcc_hosts
```

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

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

## Greenplum Command Center Users and Roles



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

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

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

**Multi-Cluster Support**

**P GREENPLUM COMMAND CENTER**

Welcome gpadmin | Pivotal | Support | Feedback | Help | Logout

gp1      Training2

gp1: Uptime:0d 0h 49m GPDB Version:4.3.4.0 build 1 Connections:1 Active Queries:0

Training2: Uptime:0d 0h 49m GPDB Version:4.3.4.0 build 1

```
# gpadmin@node01:~/local/greenplum_cc/web/intances
# MULTI-Cluster configuration file to monitor health status of each GPCC cluster. The brief explanation of following fields are as follows :
# SERVER : Server acts as primary key to uniquely identify each GPCC cluster. In UI this is used as a display name.
# If any duplicate entries exist in this column then UI throws an error message. It won't allows special characters except space, underscore(_) and hyphen(-).
# HOST : It is GPCC Hostname/ IP address to be contacted to get health information.
# PORT : The port number on which the GPCC is running.
# TABGROUP : This field is used for segregation of GPCC clusters. (like Production, Testing, Deployment, etc...).
# If no TABGROUP is specified then it is considered as an error which will be notified at UI.
# If no TABGROUP is specified in this field then a default value is set to false.
# NOTE : SERVER,HOST,PORT and TABGROUP use mandatory fields separated by colon(:)
# Example: www.oracle.com:28080:Production:true/false
# orforst:10.81.17.186:28080:Development:false:true
# Grandalipha:andalipha:32020:Development:true
# SERVER : HOST : PORT : TABGROUP : AUTOLOGIN : SSL
# gp1:10.126.88.204:28080:Production:false:true
# Training2:10.126.88.204:28081:Production:true:true
# ./gp1/conf/clusters.conf" 30L, 1775c
```

Click Multi-Cluster link after updating clusters.conf file

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

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

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

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

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

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

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

## Defining and Managing Console Instances

gpcmdr Option	Description
--setup	Configure a unique Greenplum Command Center instance
--restart [instance]	Restart Greenplum Command Center instance(s)
--start [instance]	Start Greenplum Command Center instance(s)
--stop [instance]	Stop Greenplum Command Center instance(s)
--status [instance]	Display Greenplum Command Center instance information

Instance information is stored in  
\$GPPERFMONHOME/instances/<instance\_name>

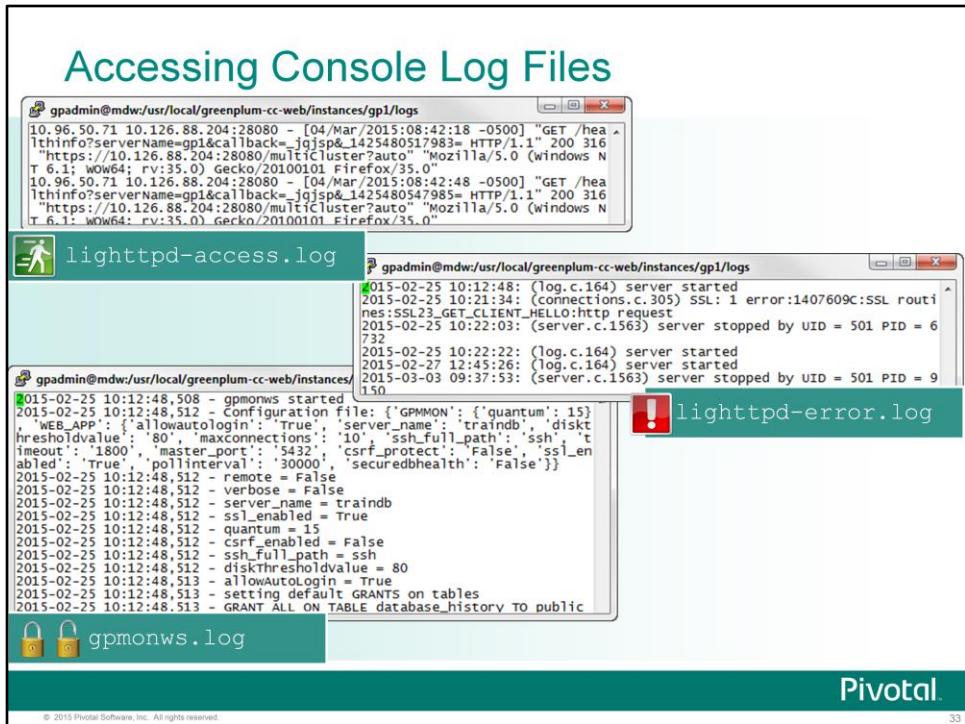
Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

32

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

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



Errors and access information are written to log files within the instance directory.

The log files, stored in

`$GPPERFMONHOME/instances/<instance_name>/logs`, are:

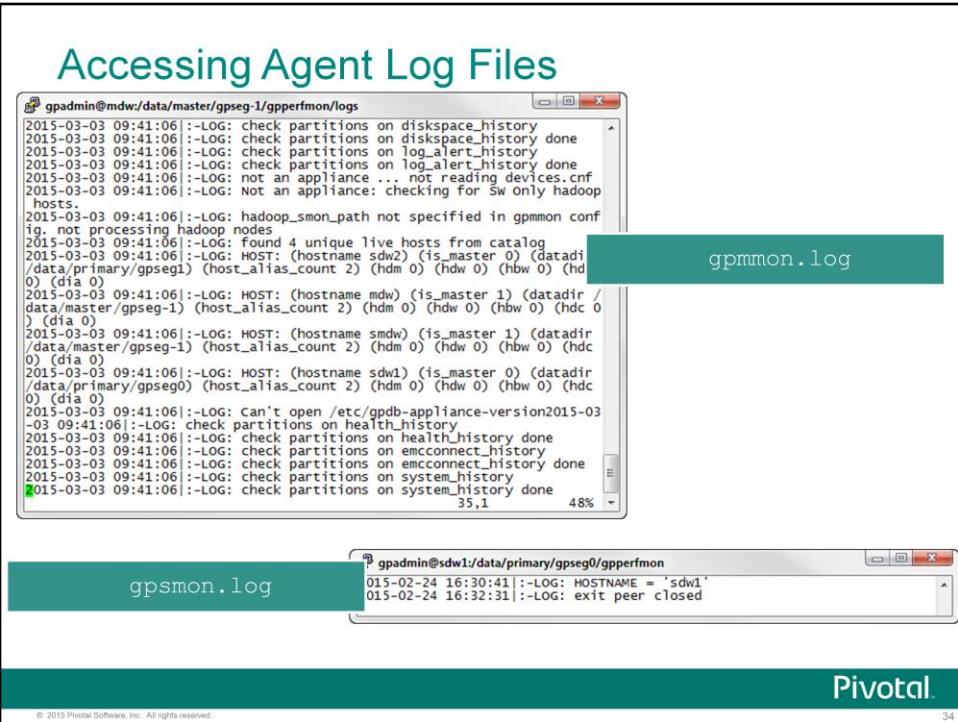
- `lighttpd-access.log` contains the access information to the instance
- `lighttpd-error.log` contains any errors for the instance

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

The file locations for the access and error log files are defined in

`$GPPERFMONHOME/instances/conf/lighttpd.conf`.

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



Log files for the master agent are stored as

`$MASTER_DATA_DIRECTORY/gpperfmon/logs/gpmon.<timestamp>.log`. This contains messages highlighting communications to other cluster agents.

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

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

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

## Lab: Install and Configure Pivotal Greenplum Command Center

In this lab, you install Greenplum Command Center and validate that it was successfully installed.

You will:

- Enable the Greenplum Command Center agents
- Install the Greenplum Command Center Console
- Configure the Greenplum Command Center Console and enable access to the console for `gpadmin`

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

35

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

## Lab: Navigating Pivotal Greenplum Command Center

In this lab, you install Greenplum Command Center and validate that it was successfully installed.

You will:

- Navigate the Pivotal Greenplum Command Center dashboards
- Obtain information on the Greenplum environment using the tabs

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

36

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

## Module 3: Greenplum Database Tools, Utilities, and Internals

### Lesson 2: Summary

During this lesson the following topics were covered:

- The purpose of Greenplum Command Center and its features and benefits
- Three main components of the Greenplum Command Center architecture
- The high-level steps required when installing and configuring the Greenplum Command Center software

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

37

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

## Module 3: Greenplum Database Tools, Utilities, and Internals

### Lesson 3: Greenplum Database Server Configuration

In this lesson, you examine the parameters that affect the behavior of the Greenplum Database system.

Upon completion of this lesson, you should be able to:

- Define segment-level and master parameters
- Set configuration parameters
- List configuration parameter categories

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

38

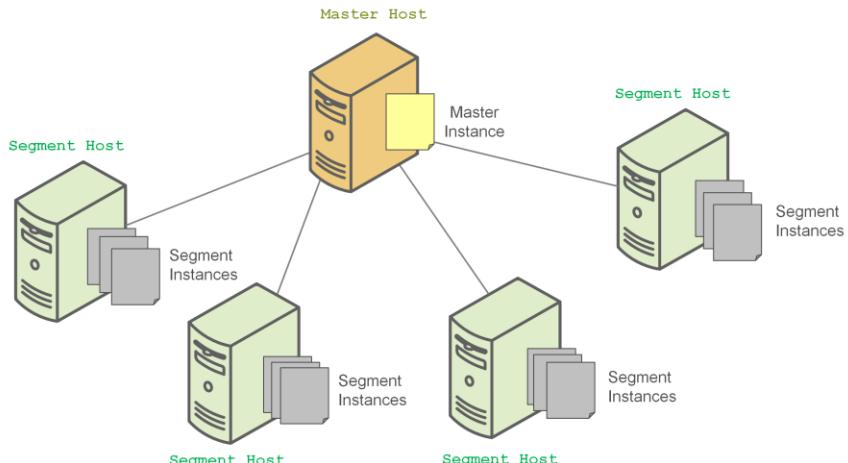
There are many server configuration parameters that affect the behavior of a Greenplum Database system, including the debug level or the behavior of write ahead logs.

In this lesson, you:

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

## Master and Segment-Level Parameters

`postgresql.conf`



Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

39

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

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

Parameters can be:

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

## Server Configuration File

The server configuration file, `postgresql.conf`,

- Is located in master and segment instance's data directory
- Contains parameters that:
  - Have limitations on how they can be changed, when, and by whom
  - Is used to set configuration parameters for the database instances running in Greenplum
  - May require a database restart or reload (`gpstop -u`) for changes to take effect
- Contains parameters that can be set at different levels:
  - System-level
  - Database-level
  - Role
  - Session-level
- Uses comments (#) in front of parameters with default settings

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

40

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

Configuration parameters may have the following restrictions:

- Some parameters have limitations on who can change them and where or when they can be set. For example, to change certain parameters, you must be a Greenplum Database superuser.
- Some parameters can only be set at the system-level in the `postgresql.conf` file.
- Other parameters may require a restart of the system for the changes to take effect.
- Many configuration parameters are considered *session* parameters. Most session parameters can be changed by any database user within their session, but a few may require superuser permissions. A session parameter can be set at the:
  - System-level
  - Database-level
  - Role-level
  - Session-level

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

## Setting Configuration Parameters

Remember the following tips when setting configuration parameters:

- Some can only be set at server start
- Runtime parameters can be set on a
  - per-user
  - per-database
  - per-session basis (user settable)
- Some require superuser permissions
- Some are read-only

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

41

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

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

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

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

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

postgresql.conf Parameter Set Classifications	
Master/Local	
Master	Parameter set on the master instance.
Local	Parameter set on the master and all segment instances.
System/Session	
System	Parameter can only be changed via the <code>postgresql.conf</code> file(s).
Session	Parameters can be changed on the fly within a database session using the <code>SET</code> SQL command.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

42

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

Parameters are defined as:

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

## postgresql.conf Parameter Set Classifications

### Restart/Reload

Restart	Database has to be brought down and back up for the parameter to change.
Reload	Run the <code>gpstop -u</code> command to re-read the <code>postgresql.conf</code> file.

### Superuser/Read Only

Superuser	Parameters can only be set by a database superuser. Regular database users cannot set this parameter.
Read Only	Parameters are not settable by database users or superusers.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

43

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

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

A read only parameter is not settable by database users.

## Set Classification Examples

Parameter	Description	Default Setting	Set Classification
max_connections	The maximum number of concurrent connections to the database server	<ul style="list-style-type: none"><li>• 250 on master instance</li><li>• 750 on segment instances</li></ul>	<ul style="list-style-type: none"><li>• Local</li><li>• System</li><li>• Restart</li></ul>
enable_hashjoin	Enables or disables the query planner's use of hash-join plan types	on	<ul style="list-style-type: none"><li>• Master</li><li>• Session</li><li>• Reload</li></ul>

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

44

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

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

## Setting Configuration Parameters

To change user-settable parameters:

- For master parameters:
  - At the system-level, edit the master `postgresql.conf`  
`log_min_messages = DEBUG1`
  - At the database-level, run the `ALTER DATABASE` command  
`ALTER DATABASE names SET search_path TO baby, public, pg_catalog;`
  - At the role-level, run the `ALTER ROLE` command  
`ALTER ROLE lab1 SET search_path TO baby, public, pg_catalog;`
  - At the session-level, run the `SET` command  
`SET search_path TO baby, public, pg_catalog;`
- For local parameters, edit each segment's `postgresql.conf`

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

45

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

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

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

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

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

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

## **Setting Configuration Parameters (continued)**

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

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

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

## Managing Parameters with gpconfig

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

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

47

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

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

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

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

## Viewing Parameter Settings

To view the value for:

- A specific parameter, run the `SHOW` command  
`SHOW search_path;`
- All parameters, run the `SHOW ALL` command  
`SHOW ALL;`
- Values for a specific parameter across the system, run the `gpconfig` utility  
`gpconfig --show max_connections`

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

48

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

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

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

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

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

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

## Configuration Parameter Categories

General categories for configuration parameters include:

- Connections, Security and Authentication
- Resource Consumption
- Query Tuning
- Error Reporting and Logging
- System Monitoring
- SNMP and Email Alerts
- Interconnect
- Fault Tolerance
- Runtime Statistics Collection
- Automatic Statistics Collection
- Client Connection
- Lock Management
- Workload Management
- External Table
- Append-only Table
- Database and Tablespace
- Greenplum Array Configuration

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

49

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

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

## Configuration Parameter Categories (Continued)

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

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

## Configuring Host-Based Authentication

Using client authentication, you determine:

- Whether connections are allowed from this client host
- Whether the user has permission to connect to the requested database

Client authentication:

- Is defined in the `pg_hba.conf` file
- Controls authentication methods for connections based on host address, database, and/or DB user account
- Is located in the master or segment instance data directories
- Has a default configuration set at system initialization time

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

51

Client authentication is the process by which the database server:

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

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

- Client host address
- Database
- User

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

## Defining the Postgresql pg\_hba.conf

- Each line is called a record
- Each record specifies
  - Connection type
  - Client IP address (if relevant for the connection type)
  - Database name(s) (comma separated)
  - User name(s) (comma separated)
  - Authentication method and options
- First match connection type is used to perform the authentication
- There is no “fall-through” or “backup”. Subsequent records are not considered.
- If no record matches, access is denied.
- Kerberos, LDAP, and PAM are supported (Require additional entries)

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

52

The general format of the pg\_hba.conf file is a set of records, one per line. Blank lines are ignored, as is any text after the # comment character. A record is made up of a number of fields which are separated by spaces and/or tabs. Fields can contain white space if the field value is quoted. Records cannot be continued across lines.

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

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

## pg\_hba.conf Record Examples

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

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

53

### Example 1:

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

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

### Example 2:

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

### Example 3:

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

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

### Example 4:

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

## **pg\_hba.conf Record Examples (continued)**

Example 5:

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

Example 6:

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

The file \$MASTER\_DATA\_DIRECTORY/admins contains a list of names of administrators. Passwords are required in all cases.

Example 7:

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

Passwords are NOT required.

Example 8:

Allow any user from any host with IP address 192.168.93.x to connect to databases "db1", "db2", and databases found in the \$MASTER\_DATA\_DIRECTORY/listdb file.

Passwords are required in all cases.

## Default Segment Host Client Authentication

The following is the setting for client authentication in a segment's pg\_hba.conf file.

```
# TYPE      DATABASE   USER      CIDR-ADDRESS     METHOD
local      all        all       trust
host       all        all       127.0.0.1/24    trust
host       all        all       ::1/128        trust
host       all        all       ::1/32         trust
host       all        all       <localaddr>/32  trust
host       all        all       fe80::.../32    trust
...
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

55

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

Let us examine each line of the pg\_hba.conf file for the segment hosts and explain what is happening:

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

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

## Default Master Host Client Authentication

The following is the default setting for client authentication in the master host's pg\_hba.conf file.

#	TYPE	DATABASE	USER	CIDR-ADDRESS	METHOD
	local	all	gpadmin	ident	
	host	all	gpadmin	127.0.0.1/28	trust
	host	all	gpadmin	<localaddr>/32	trust
	host	all	gpadmin	::1/128	trust
	host	all	gpadmin	fe80::.../128	trust

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

56

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

Let us examine each line of the pg\_hba.conf file for the master host and explain what is happening:

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

## Lab: Greenplum Database Server Configuration

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

You will:

- Set configuration parameters in `postgresql.conf` and use the `SHOW` command in `psql` to examine the current values of configuration parameters
- Set Greenplum global parameters on a per-session basis using the `SET` command in `psql`
- Set and view parameters using the `gpconfig` command

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

57

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

## Module 3: Greenplum Database Tools, Utilities, and Internals

### Lesson 3: Summary

During this lesson the following topics were covered:

- Local and master parameters
- Configuration parameters
- Configuration parameter categories

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

58

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

## Module 3: Greenplum Database Tools, Utilities, and Internals

### Lesson 4: Greenplum Database Internals

In this lesson, you examine the system catalog, the physical storage of Greenplum Database and its objects, and the server processes.

Upon completion of this lesson, you should be able to:

- Describe system catalog tables
- Define physical storage
- Describe server processes

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

59

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

In this lesson, you will:

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

## System Catalog Tables and Views

List of relations				
Schema	Name	Type	Owner	Storage
pg_catalog	gp_configuration	table	gpadmin	heap
pg_catalog	gp_configuration_history	table	gpadmin	heap
pg_catalog	gp_db_interfaces	table	gpadmin	heap
pg_catalog	gp_distributed_log	table	gpadmin	heap
pg_catalog	gp_distributed_xad	table	gpadmin	heap
pg_catalog	gp_distribution_pd	table	gpadmin	heap
pg_catalog	gp_fastsequence	table	gpadmin	heap
pg_catalog	gp_fault_strategy	table	gpadmin	heap
pg_catalog	gp_global_sequence	table	gpadmin	heap
pg_catalog	gp_id	table	gpadmin	heap
pg_catalog	gp_interfaces	table	gpadmin	heap
pg_catalog	gp_persistent_database_node	table	gpadmin	heap
pg_catalog	gp_persistent_filespace_node	table	gpadmin	heap
pg_catalog	gp_persistent_relation_node	table	gpadmin	heap
pg_catalog	gp_persistent_tablespace_node	table	gpadmin	heap
pg_catalog	gp_pgdatabase	view	gpadmin	none
pg_catalog	gp_relation_node	table	gpadmin	heap
pg_catalog	gp_san_configuration	table	gpadmin	heap
pg_catalog	gp_segment_configuration	table	gpadmin	heap
pg_catalog	gp_transaction_log	table	gpadmin	heap
pg_catalog	gp_verification_history	table	gpadmin	heap
pg_catalog	gp_version_at_initdb	table	gpadmin	heap
pg_catalog	pg_aggregate	function	gpadmin	heap
pg_catalog	pg_am	operator	gpadmin	heap
pg_catalog	pg_amop	operator	gpadmin	heap
pg_catalog	pg_amproc	operator	gpadmin	heap

Note pg\_ or gp\_ precedes table and view names.  
pg\_ represents PostgreSQL standard relations as well as Greenplum tables and views related to features added to enhance PostgreSQL

The following PSQL meta commands lets you view this list:  
\dtS : List system tables  
\dtv: List system views  
\dS: List system views and tables

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

60

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

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

Greenplum uses the standard PostgreSQL system catalog tables, such as pg\_database, pg\_inherits, pg\_index, and pg\_class.

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

## Greenplum Specific Tables and Views

The screenshot shows a terminal window titled "gpadmin@mdw:/data/master/gpseg-1". It displays two command outputs:

```
gpadmin=# \dt pg_catalog(gp_*)  
List of relations  
Schema | Name | Type | Owner | Storage  
-----+-----+-----+-----+-----  
pg_catalog | gp_configuration | table | gpadmin | heap  
pg_catalog | gp_configuration_history | table | gpadmin | heap  
pg_catalog | gp_db_interfaces | table | gpadmin | heap  
pg_catalog | gp_distribution_policy | table | gpadmin | heap  
pg_catalog | gp_fastsequence | table | gpadmin | heap  
pg_catalog | gp_fault_strategy | table | gpadmin | heap  
pg_catalog | gp_global_sequence | table | gpadmin | heap  
pg_catalog | gp_id | table | gpadmin | heap  
pg_catalog | gp_interfaces | table | gpadmin | heap  
pg_catalog | gp_persistent_database_node | table | gpadmin | heap  
pg_catalog | gp_persistent_filespace_node | table | gpadmin | heap  
pg_catalog | gp_persistent_relation_node | table | gpadmin | heap  
pg_catalog | gp_persistent_tablespaces_node | table | gpadmin | heap  
pg_catalog | gp_relation_node | table | gpadmin | heap  
pg_catalog | gp_san_configuration | table | gpadmin | heap  
pg_catalog | gp_segment_configuration | table | gpadmin | heap  
pg_catalog | gp_verification_history | table | gpadmin | heap  
pg_catalog | gp_version_at_initdb | table | gpadmin | heap  
(18 rows)  
  
gpadmin=# \dv pg_catalog(gp_*)  
List of relations  
Schema | Name | Type | Owner | Storage  
-----+-----+-----+-----+-----  
pg_catalog | gp_distributed_log | view | gpadmin | none  
pg_catalog | gp_distributed_xacts | view | gpadmin | none  
pg_catalog | gp_pgdatabase | view | gpadmin | none  
pg_catalog | gp_transaction_log | view | gpadmin | none  
(4 rows)
```

A callout box highlights the first table listed in the first output:

Tables preceded by gp\_ are related to the parallel features of the Greenplum Database.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

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

- **gp\_segment\_configuration** – This contains the GPDB array master host and segment instances. It provides information on the status, mode, hostname, address, replication port, and whether they are in SAN instead of on the local environment.
- **gp\_distribution\_policy** – GPDB distribution key columns for a table
- **gp\_id** – Each segment has a local copy of this table with its own dbid and contentid
- **gp\_version\_at\_initdb** – This table is populated on the master and each segment in the Greenplum Database system. It identifies the version of Greenplum Database used when the system was first initialized.
- **pg\_resqueue** – Populated only on the master, this table contains information about Greenplum Database resource queues, which are used for the workload management feature.
- **pg\_exttable** – This table is used to track external tables and web tables created by the CREATE EXTERNAL TABLE command.

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

## Statistics Collector

The statistics collector:

- Collects information about database activity
- Counts accesses to tables and indexes
- Can add overhead to queries
- Uses the server configuration parameters:
  - start\_stats\_collector = on
  - stats\_block\_level = off
  - stats\_row\_level = off
  - stats\_queue\_level = off
  - stats\_command\_string = on

To see statistics views and tables in catalog, use the meta-command, \dtvS pg\_stat\*

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

62

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

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

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

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

## Statistics Collector – Monitoring Current Activity

When monitoring activity, note:

- Information does not update instantaneously
- Each server process transmits new block and row access counts before becoming idle
- The collector creates a new report at most every 500 milliseconds
- Current query information is always up to date

To see statistics views and tables in catalog, use the meta-command, `\dtvS pg_stat*`

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

63

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

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

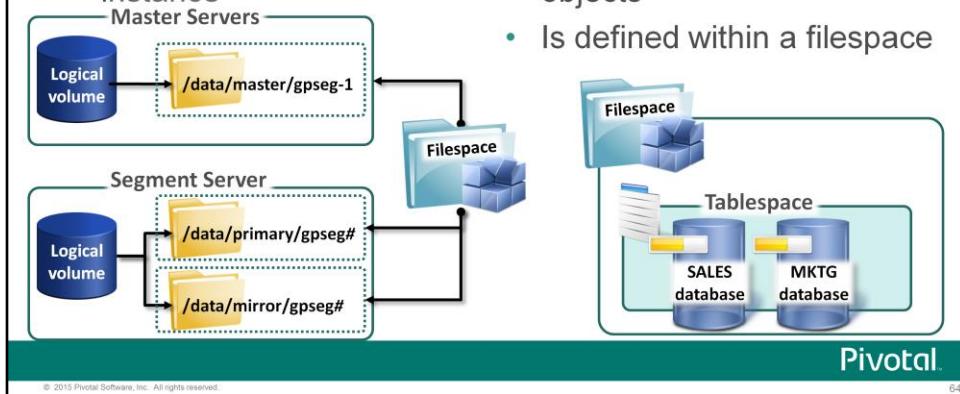
- The information does not update instantaneously.
- Each individual server process transmits new block and row access counts to the collector just before going idle; so a query or transaction still in progress does not affect the displayed totals.
- The collector emits a new report at most once per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

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

## Filespace and Tablespace

A filesystem:

- Has a distinct storage location
- Is required for each master and segment instance



A tablespace:

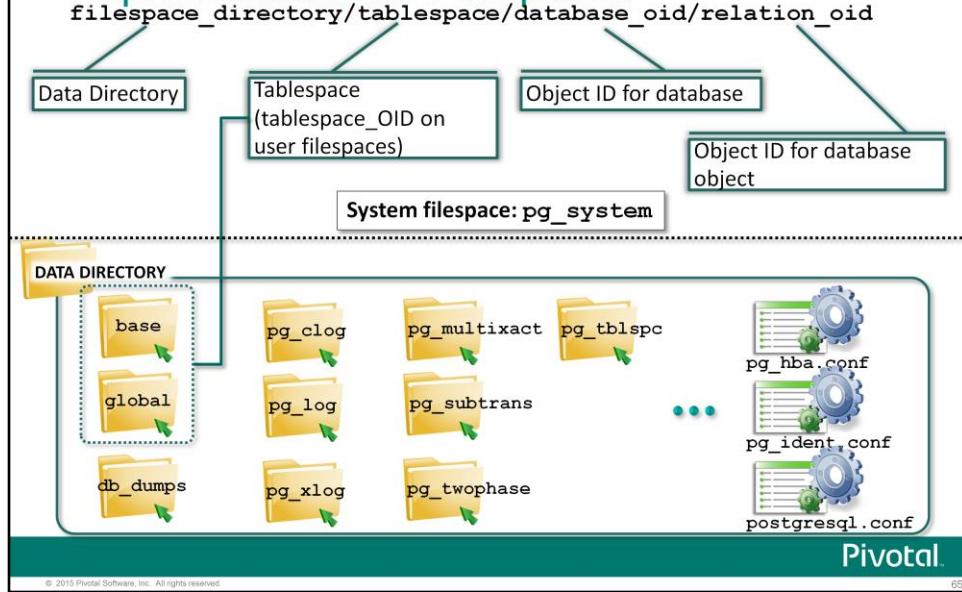
- Lets you define how best to use physical storage for database objects
- Segregates database objects
- Is defined within a filesystem

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

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

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

## Physical Storage – Physical Data Representation for Filespaces

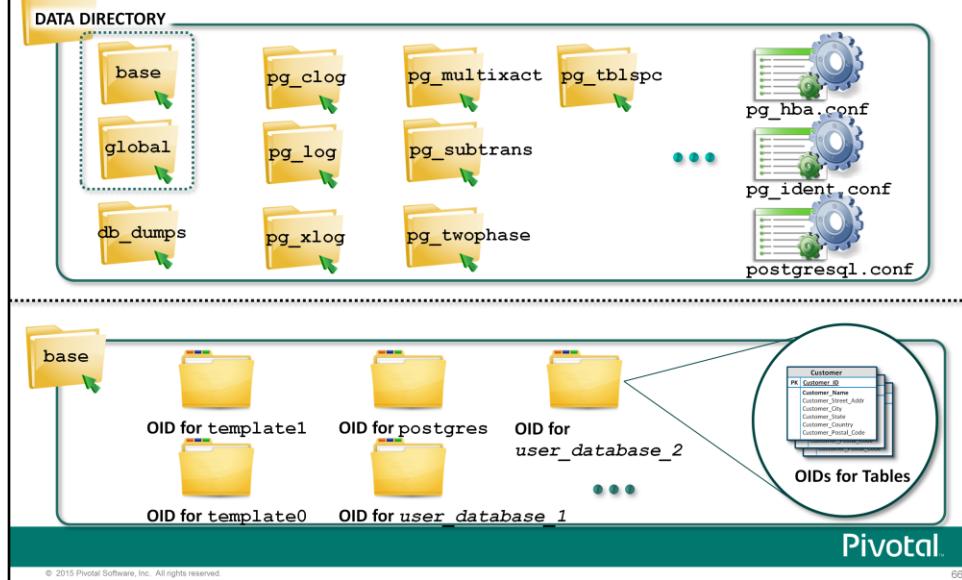


Data is represented on the file system with file structures that use object identifiers, OID. Each OID is a unique identifier that describes that object. Data is stored in the file structure, `filespace_directory/tablespace/database_oid/relation_oid`.

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

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

## Physical Storage – Physical Data Representation for Filespaces (Cont)



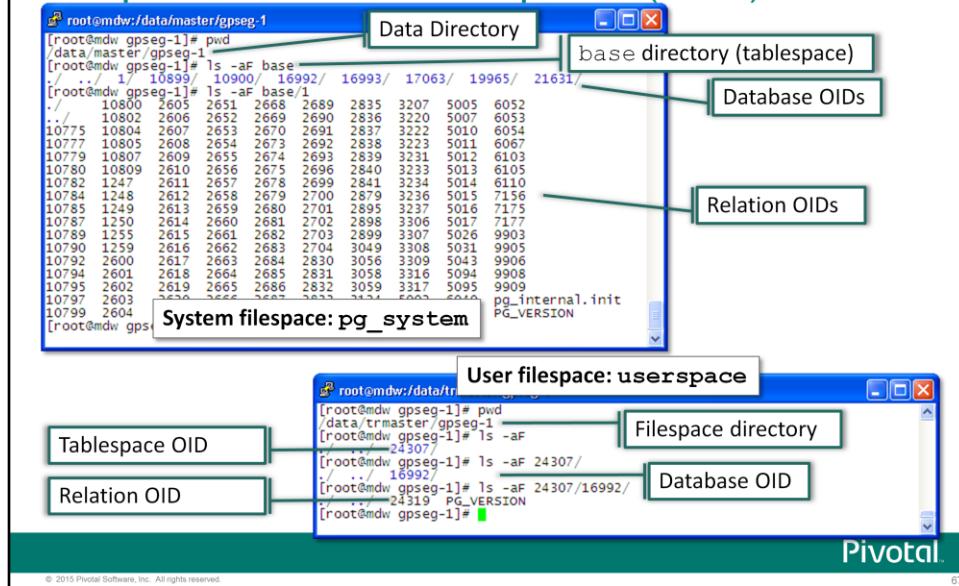
Under each data directory for the system filesystem, the file structure for the Greenplum Database is as follows:

- **base** – This directory contains subdirectories for each database, identified by their object identifier (OID) in pg\_database. Each sub-directory contains a list of the tables, stored as files. These files are identified by their OID as defined in pg\_class.
- **global** – The global directory, contains system-wide tables, such as pg\_database and gp\_configuration. The database objects are stored directly in the global directory, instead of in subdirectory representing a database OID.
- **pg\_clog** – This directory contains transaction commit status data.
- **pg\_multixact** - This directory stores multitransaction status data, used for shared row locks.
- **pg\_subtrans** – This directory stores subtransaction status data.
- **pg\_tblspc** – This directory contains symbolic links to tablespaces.
- **pg\_twophase** – The state files for prepared transactions are stored here.
- **pg\_xlog** – Write Ahead Log (WAL) files are stored in this directory.

Files stored in this structure include:

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

## Physical Storage – Physical Data Representation for Filespaces (Cont)



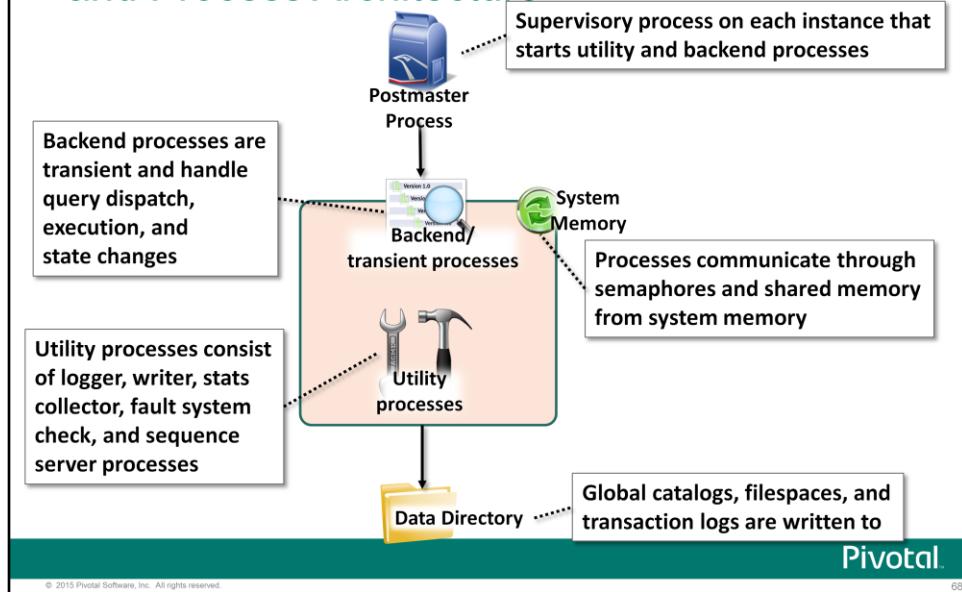
This slide shows how the file structure is represented on a populated system. The first image shows the base directory located inside the data directory on the master and segment instances. Within the base directory are the database OIDs representing different database names that exist on the system,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

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

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

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

## Greenplum Master and Segment Memory and Process Architecture



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

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

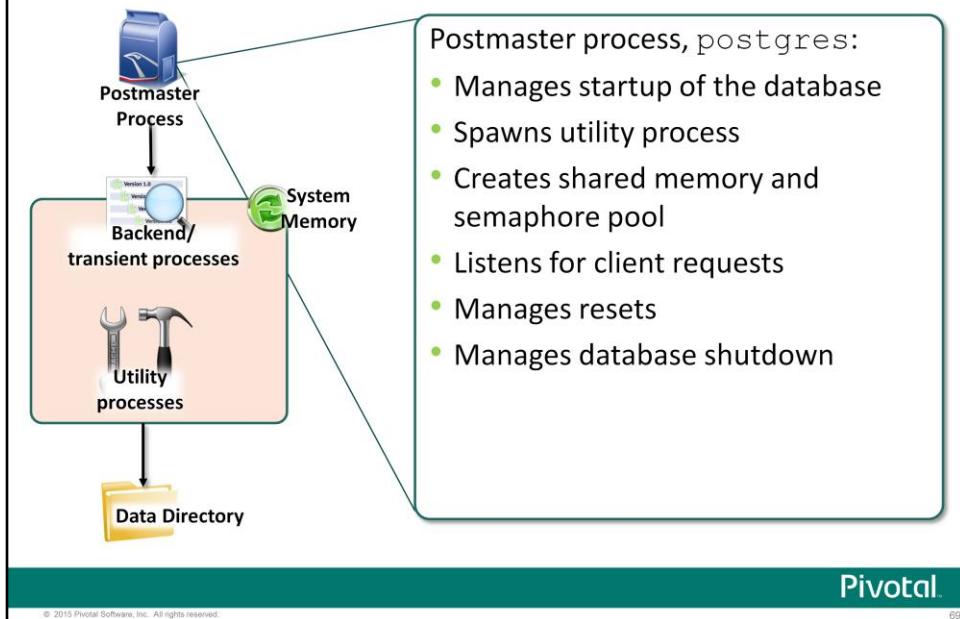
Each primary and segment instance has a single supervisory process called the postmaster, which is responsible for spawning and managing processes associated with starting and stopping the database, starting and managing backend and utility processes, and creating shared memory.

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

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

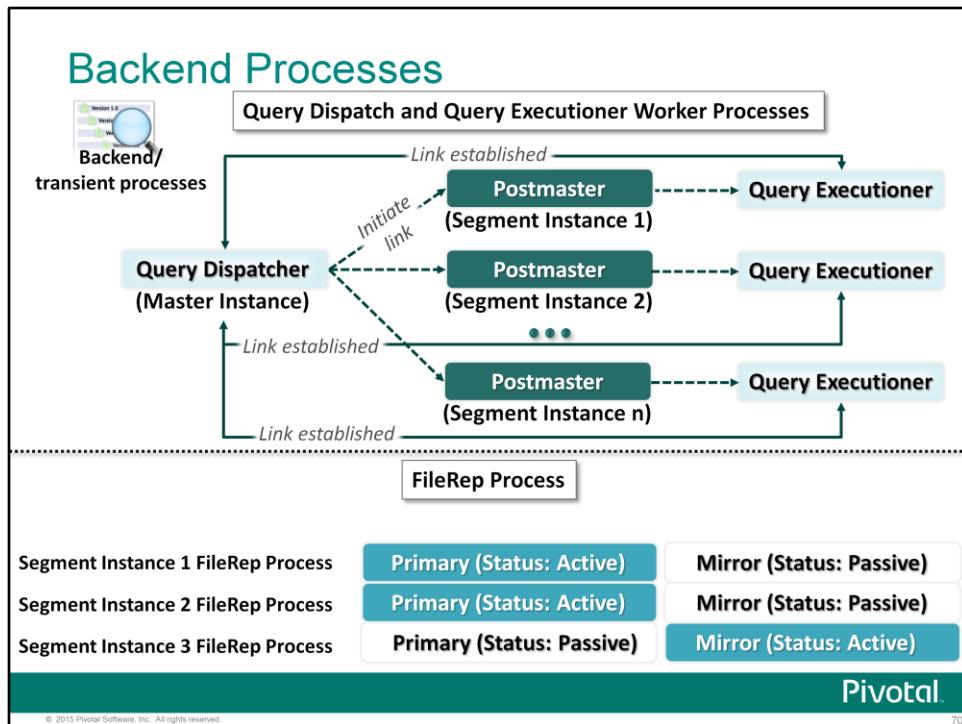
These processes all communicate through shared memory created by the postmaster.

## Postmaster Process



The postmaster process, `postgres`, is responsible for:

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



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

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

## Utility Processes



Logger  
Process

- Logs activity details to pg\_log
- Sends email alerts, if configured



Writer Process

- Write dirty blocks to disk when:
- Checkpoint or checkpoint timeout occurs
  - No space left in buffer



Stats Collector  
Process

- Records table and index accesses
- Submit relevant information to collector



ftsprobe  
Process

- Fault tolerant service
- Periodic health checks of instances



seqserver  
Process

- Sequential server
- Generate globally unique sequence number

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

71

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

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

## Listing Server Processes

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

```
ps ax | grep postgres
```

The `postgres` database listener process:

- On the Greenplum master instance includes:

```
postgress postmaster process  
postgres: <port port#>, <sub_process_name>  
postgres: <port port#> <user> <database> <con#> <host>  
<cmd#><slice#>
```

- On the Greenplum segment instance includes:

```
postgres postmaster process (primary)  
postgres postmaster process (mirror)  
process postgres: <sub_process_name>
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

72

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

## Lab: Database Internals

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

You will:

- 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
- Examine the database server processes running on a Greenplum Database host

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

73

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

## Module 3: Greenplum Database Tools, Utilities, and Internals

### Lesson 4: Summary

During this lesson the following topics were covered:

- System catalog tables
- Physical storage
- Server processes

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

74

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

## Module 3: Summary

Key points covered in this module:

- Using the PSQL client to connect to the Greenplum Database and access database objects
- Installing and configuring Greenplum Command Center to view the system and Greenplum Database health
- Viewing and setting configuration parameters used to configure Greenplum Database instances and authentication controls used to determine access to the Greenplum Database instances
- Listing the system catalog stored on the master server and examine the physical file structure and processes associated with the database

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

75

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

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

This slide is intentionally left blank.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

76

# Module 4: Defining and Securing the User Database

This module examines the Data Definition Language used to manage database objects, roles, privileges, and resource queues. It also examines how to implement security measures to protect data and balance workloads.

Upon completion of this module, you should be able to:

- Identify and manage database objects available in the Greenplum Database
- Use data manipulation language and data query language to access, manage, and query data
- Manage workload management processes by defining and managing roles, privileges, and resource queues
- Implement system-level and database-level security

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

1

Data definition language (DDL) defines the data structures you will manage within the Greenplum Database. You use SQL syntax to create, alter, and remove database objects, including tables, schemas, and views.

As an administrator or developer, you must also learn the SQL syntax to manage the workload resources, which includes managing roles, privileges, and the resource queues. Once you know how to manage these items using SQL syntax, you can implement security measures to protect data and resources from unauthorized users.

In this module, you will:

- Identify and manage database objects for the Greenplum Database.
- Use data manipulation language and data query language to access, manage, and query data
- Manage workload management processes by defining and managing roles, privileges, and resource queues.
- Implement system-level and database-level security.

## Module 4: Defining and Securing the User Database

### Lesson 1: Data Definition Language

In this lesson, you examine the data structures and the syntax to manage data objects in the Greenplum Database.

Upon completion of this lesson, you should be able to:

- Describe databases
- Describe schemas and tables
- Identify constraints
- Describe data types and constants
- Describe views, indexes, sequences, triggers, and tablespaces

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

2

Data Definition Language is the SQL syntax that you use to manage database objects. In this lesson, you examine the data structures and the syntax for managing database objects in Greenplum.

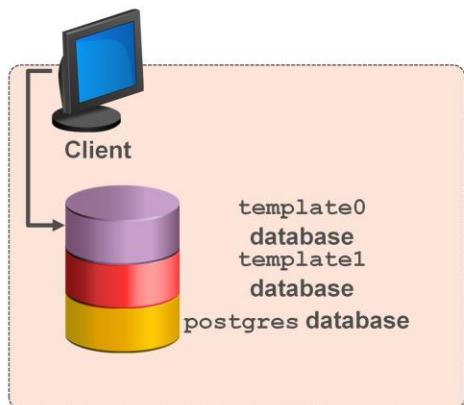
Once you have completed this lesson, you should be able to:

- Describe what a database is and how to manage it with SQL commands.
- Describe schemas and tables and manage these with SQL commands.
- Identify constraints.
- Describe data types and constants in the Greenplum Database.
- Describe other database objects, including views, indexes, sequences, triggers, and tablespaces.

## Databases – Overview

Greenplum Database instance:

- Supports multiple databases
- Creates three databases by default:
  - template0
  - template1
  - postgres



Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

3

As in PostgreSQL, an Greenplum Database instance can have one or more user created databases. This differs from systems such as Oracle where the DBMS and database instance are one in the same. These databases reside on a single system but do not share data.

Clients can connect to and have access to one database at a time.

Three databases are created by default in a newly initialized array:

- **template1** – This default database is the template used to create other databases. Do not create any objects in this database unless you want those objects to also be in every other database you create moving forward.  
Any database can be used as a template. A template provides you with a method of cloning databases and carrying specific base objects forward. Use the SQL syntax, `CREATE DATABASE newname USING TEMPLATE clonename` to create a database based on a template.
- **template0** – This template is a blank template used to create template1. This should not be dropped or modified.
- **postgres** – This is a default PostgreSQL database introduced in PostgreSQL 8.1 and is used internally by the system. It must not be dropped or modified.

## Database SQL Commands

To manage databases, use the following SQL syntax or Greenplum application:

Action	SQL Syntax	Greenplum Application
Create a database	CREATE DATABASE	createdb
Drop a database	DROP DATABASE	dropdb
Alter a database: • Rename the database • Assign a new owner • Set configuration parameters	ALTER DATABASE	

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

4

PostgreSQL and Greenplum Database share the same SQL commands for creating the database. You can use either SQL syntax in a PostgreSQL client to create, drop, or alter the database or use the Greenplum application clients available to create and drop the database.

## PSQL Database Tips

Here are some tips for working in PSQL:

- PSQL prompt shows which database you are in  
**EXAMPLE:** template1=# (superuser)  
                  names=> (non-superuser)
- \? To obtain a list of PSQL meta-commands
- \h to obtain a list of SQL commands or additional help on the included command
- To show a list of all databases:  
\l
- To connect to another database:  
\c db\_name



**Note:** Use PGDATABASE environment variable to set the default database

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

5

Here are some helpful tips for working with databases in the PSQL client program:

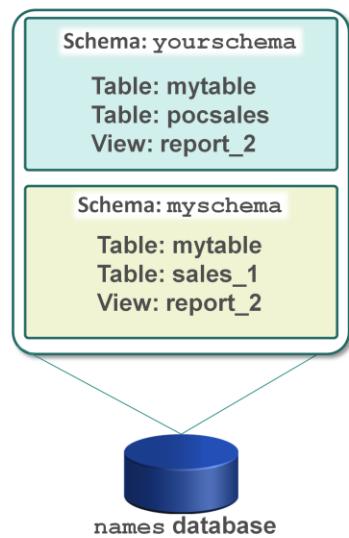
- The psql prompt will show the name of the database you are connected to and if you are in as a superuser role or not. For this class, we are logged in as a superuser.
- You can obtain additional assistance with the list of available meta-commands with \?.
- \h provides a list of SQL commands and lets you obtain additional assistance on a specific command if the command is provided as part of the command line.
- \l shows a list of all databases, including the templates and defaults databases created.
- Use \c to connect to another database or to reconnect to the same database. For example, after setting a parameter on a database you will need to reconnect to the database to see the changes.

**Note:** If you have set the PGDATABASE variable in your shell, you do not have to type the database name when connecting with psql.

## Schemas – Overview

Schemas:

- Logically organize objects
- Do not represent users
- Contain data objects
- Use *qualified* names to access objects  
EXAMPLE:  
myschema.mytable
- Can be added to the search path (search\_path)



Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

6

Schemas:

- Are a way to logically organize a database for users or for functional areas. For example, data from third-party applications can be stored in separate schemas to prevent collision with the names of other objects.
- Do not represent users, as is seen in Oracle.
- Contain named objects such as tables, views, indexes, data types, functions, and operators.

Allow you to have more than one object, such as tables, with the same name in the database without conflict, as long as they are in different schemas.

For the database to know in which schema it should look for an object, you use the *qualified* name. If you do not want to type the qualified name all the time, you can set the `search_path` parameter. This tells the database in which order it should search the available schemas for objects. The schema listed first in the search path becomes the default schema, which is where new, unqualified objects will be created by default.

As a best practice, you should always specify a schema when allowed. Changes in data with the addition of more objects with the same name in different schemas may cause issues when the optimizer and the query attempts to resolve table names.

Unlike databases, schemas are not rigidly separated - a user can access objects in any of the schemas in the database he is connected to, if he has privileges to do so.

## Schemas and the Search Path

To access the `search_path` parameter:

- Run the following PSQL command to set the parameter:

```
SET search_path TO myschema,public;
```

- Run the following command to view the value of the parameter:

```
SHOW search_path;  
search_path  
-----  
myschema,public
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

7

PostgreSQL will search for tables that are named without qualification by using the `search_path` parameter. Tables without qualifications are tables that do not have the schema name specified when calling the table name.

The following is used to access the `search_path` parameter from PSQL:

- To set the `search_path` parameter, issue the following command:  
`SET search_path TO myschema,public;`
- To view the contents of the `search_path` parameter, issue the following command:  
`SHOW search_path;`

## Pre-existing Schemas

The following schemas exist in every database:

- PUBLIC schema
- System level schemas:

Schema	Description
pg_catalog	Stores system catalog names, functions, and operators
information_schema	Contains views that describe objects in the database
pg_toast	Stores large objects that exceed page size
pg_bitmapindex	Stores bitmap index objects
pg_aoseg	Stores append-only objects
gp_toolkit	Administrative schema

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

8

Every newly created database has a default schema named PUBLIC. By default, all roles have CREATE and USAGE privileges on the public schema. Any other schemas you create you will have to grant the appropriate privileges to roles other than the owner.

The following system-level schemas also exist in every database:

- **pg\_catalog** is the schema that has the system catalog tables, built-in data types, functions, and operators. It is always part of the search path, even if it is not named.
- **information\_schema** consists of a set of views that contain information about the objects in the database. This is used by psql meta-commands to get information from the system catalog tables.
- **pg\_toast** are where large objects are stored, such as records that exceed the page size.
- **pg\_bitmapindex** where bitmap index objects are stored (list of values, etc).
- **pg\_aoseg** is the schema that stores append-only objects.
- **gp\_toolkit** is the administrative schema that you use to view and query system log files and other system metrics. It contains a number of external tables, views, and functions that you can access with SQL commands and is accessible by all database users.

## Schema SQL Commands

To manage schemas, use the following SQL syntax:

Action	SQL Syntax
Create a schema	CREATE SCHEMA
Drop a schema	DROP SCHEMA
Alter a schema: • Change name • Assign new owner	ALTER SCHEMA

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

9

PostgreSQL and Greenplum Database share the same SQL commands for creating and managing schemas.

## PSQL Schema Tips

Here are some tips for working in PSQL:

- To see the current schema, run the following command:  
`SELECT current_schema();`
- To see a list of all schemas in the database, run the following:  
`\dn`
- To see the schema search path:  
`SHOW search_path;`
- To set the search path for a database:  
`ALTER DATABASE SET search_path TO myschema, public, pg_catalog;`

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

10

Here are some helpful tips for working with databases in the PSQL client program.

## Tables – Overview

### Tables in relational databases:

- Consist of columns and rows
- Have a fixed number and order of named columns
- Has a variable number of rows reflecting individual records
- No guaranteed row order

### Tables share the following characteristics:

- All tables in Greenplum Database are distributed based on a distribution key
- Some table features are not yet supported in Greenplum Database
  - Foreign key constraints
  - Limit on unique constraints

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

11

A table in a relational database:

- Is much like a table on paper where it consists of columns and rows.
- Has a fixed number and order of columns and each column is named.
- Has a variable number of rows reflecting how much data is stored at a given moment.

PostgreSQL makes no guarantees about the order of the rows in a table.

One important difference about tables in Greenplum Database is that all tables are distributed, meaning they are divided into parts amongst the primary segment instances. The distribution is done using a sophisticated proprietary hashing algorithm on the distribution key of the table.

The distribution key can be declared when the table is created or altered. It should be chosen with care.

Because tables are distributed, some features are not yet supported Greenplum Database:

- Foreign key integrity cannot yet be enforced in a distributed table, so this is not supported.
- A table cannot have multiple columns with unique constraints, and any single column that has a unique constraint must also be declared as the distribution key.

## Table Distribution Keys – Overview

When defining distribution keys:

- One or more columns are used to divide the data among the segments
- They should be unique to ensure even data distribution
- Specified with `CREATE TABLE` or `ALTER TABLE` using the `DISTRIBUTED BY` clause

```
CREATE TABLE sales
(dt date, prc float, qty int, cust_id int,
 prod_id int, vend_id int)
DISTRIBUTED BY (dt, cust_id, prod_id);
```

- If table does not have a unique column or set of columns, declare `DISTRIBUTED RANDOMLY` as the distribution key
- If not explicitly declared, defaults to the table's `PRIMARY KEY` or the first column of the table

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

12

Let us examine distribution keys in greater detail, as this is a very important concept in Greenplum Database. Table distribution is critical to getting the best performance possible.

- In a table, one or more columns are used as the distribution key, meaning those columns are used to divide the data amongst all of the segments.
- To ensure an even distribution of data, you want to choose a distribution key that is unique for each record. If that is not possible, declare `DISTRIBUTED RANDOMLY` as the distribution key.
- The distribution key is chosen when the table is created or altered using the `DISTRIBUTED BY` clause. This is a Greenplum extension to the SQL standard. Note that you cannot alter the columns of a table that are used as the distribution key after the table is created.
- If a `DISTRIBUTED BY` clause is not supplied, then either the `PRIMARY KEY`, if the table has one, or the first eligible column of the table will be used as the distribution key. This may or may not be the desirable distribution key. Columns of geometric data types are not eligible as distribution key columns. If a table does not have a column of an eligible data type, the rows are distributed based on a random distribution.

## Table SQL Commands

To manage tables, use the following SQL syntax:

Action	SQL Syntax
Create a table	CREATE TABLE
Drop a table	DROP TABLE
Alter a table	ALTER TABLE

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

13

PostgreSQL and Greenplum Database share the same SQL commands for creating and managing tables.

## Defining the Default Distribution Method

gp_create_table_random_default_distribution	
Value	Description
OFF	(Default) The table distribution key is based on the CREATE TABLE/CREATE TABLE AS command
ON	When the DISTRIBUTED BY clause is not specified: <ul style="list-style-type: none"><li>Primary key: Not specified Random distribution</li><li>Unique column: Not specified</li></ul> Primary or Unique key: Specified DISTRIBUTED BY clause must be included

**Usage: Update default storage options at role level**

```
$ gpconfig -c gp_create_table_random_default_distribution -v 'ON'  
$ gpstop -u
```

**gpadmin@mdw:~**

```
names=> create table db_lvl32 (nid int, strm text);  
NOTICE: Using default RANDOM distribution since no distribution was specified.  
HINT: Consider including the 'DISTRIBUTED BY' clause to determine the distribution of r  
ows.  
CREATE TABLE  
names=>
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

The `gp_create_table_random_default_distribution` parameter defines how the `CREATE TABLE` and `CREATE TABLE AS` statements behave with regards to table distribution.

When the parameter is set to `OFF`, which is the default value, the table distribution key follows the methods outlined earlier. If the `DISTRIBUTED BY` clause is not specified:

- The primary key is used, if the table has one, or the first eligible column is used.
- The table inherits the parent table or source table if the `LIKE/INHERITS` clause is used

If the parameter is set to `ON`:

- And neither the primary key or a unique column is defined, random distribution is chosen.
- Either the primary key or the unique column is defined, the `DISTRIBUTED BY` clause must be included in the `CREATE TABLE` command or the command will fail.

The parameter is settable with the `gpconfig` command and requires a reload if set.

## Table SQL Commands – Creating a Table

The following is an example of creating a table:

### Example: Creating a Table

```
CREATE TABLE dimensions.store
(
    storeId      SMALLINT      NOT NULL,
    storeName    VARCHAR(40)   NOT NULL,
    address      VARCHAR(50)   NOT NULL,
    city         VARCHAR(40)   NOT NULL,
    state        CHAR(2)       NOT NULL,
    zipcode      CHAR(8)       NOT NULL
);
```

Pivotal

In this example, the table, `store`, is created in the schema, `dimensions`. The table consists of 6 columns of varying data types.

## Table SQL Commands – Modifying a Table

You can modify certain attributes of a table, including:

- Renaming columns
- Renaming tables
- Adding and removing columns

```
ALTER TABLE product ADD COLUMN description text;
```

- Adding and removing constraints

```
ALTER TABLE product ALTER COLUMN prod_no SET NOT NULL;
```

- Changing default values

```
ALTER TABLE product ALTER COLUMN prod_no SET DEFAULT -  
999;
```

- Changing column data types

```
ALTER TABLE products ALTER COLUMN price TYPE  
numeric(10,2);
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

16

You can alter several properties of a table, including:

- Column names
- Table name
- The number of columns by adding or removing columns
- Constraints on the table by adding or removing constraints
- Default values for columns in the table
- Column data types

## Table SQL Commands – Removing a Table

To remove a table:

- Issue the DROP TABLE command:  
`DROP TABLE dimensions.customer_old;`
- And prevent an error should the table not exist, use IF EXISTS in the DROP TABLE statement  
`DROP TABLE IF EXISTS dimensions.customer_old;`

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

17

When removing the table, you should qualify the table name and specify the name of the table to be dropped. If the table does not exist, you can avoid error messages by using IF EXISTS in the DROP TABLE statement.

## Table Column Basics

In PostgreSQL and Greenplum Database:

- Each column has a data type
- There is a large set of data types available
- You can define named data types
- There is a limit on the number of columns in a table

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

18

With PostgreSQL and the Greenplum Database, there are some criteria to column definition you should be aware of:

- Each column has a data type.
- PostgreSQL includes a sizable set of built-in data types that fit many applications.
- Users can define their own data types.
- There is a limit on the number of columns a table can contain. Depending on the column types, it is between 250 and 1600. However, defining a table with anywhere near this many columns is highly unusual and often a questionable design.

The limit is due to the optimizer. An amount greater than 1600 columns will impact performance.

## Table and Column Constraints – Overview

Table and column constraints are supported with some limitations:

- CHECK table or column constraints
- NOT NULL column constraints
- UNIQUE column constraints
  - Only one constraint allowed per table
  - Unique columns must also be in distribution key
  - Not allowed if table also has a primary key
- PRIMARY KEY is used as the distribution key by default
- FOREIGN KEY constraints not supported – *referential integrity* is not enforced

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

19

Constraints are a way to limit and control the data that a table contains, limiting the data to a set of valid values. PostgreSQL supports these table and column constraints as does Greenplum Database, with some exceptions:

- CHECK constraints limit data based on some condition you define.
- NOT NULL constraints verify that a column does not have a null value when a row is created. A way of forcing users to define a value for a column when a new record is created.
- A primary key indicates that a column or group of columns can be used as a unique identifier for rows in a table. Technically, a PRIMARY KEY constraint is simply a combination of a UNIQUE constraint and a NOT NULL constraint. If a table has a primary key, it will be chosen as the Greenplum distribution key by default. The only way Greenplum Database can enforce uniqueness is if a column is also in the distribution key. Therefore, a table cannot have more than one UNIQUE constraint and any UNIQUE column or set of columns must be in the distribution key.
- Also PostgreSQL and Greenplum Database always implicitly creates a pkey index for a table's primary key.
- FOREIGN KEYS, while they can be defined, are not currently supported in Greenplum Database, as there is currently no way to enforce referential integrity between the segment instances.

## Table and Column Constraints

### Example: Table and Column Constraints

```
CREATE TABLE TR_CONSTRAINTS (
    transactionid int NOT NULL UNIQUE,
    price numeric CHECK (price > 0),
    on_sale char DEFAULT 'n'
);
```

Column cannot be NULL

Data in the column for that table is unique

If no value is entered, a default value of 'n' is entered for that field

Column cannot have a value equal to or less than zero

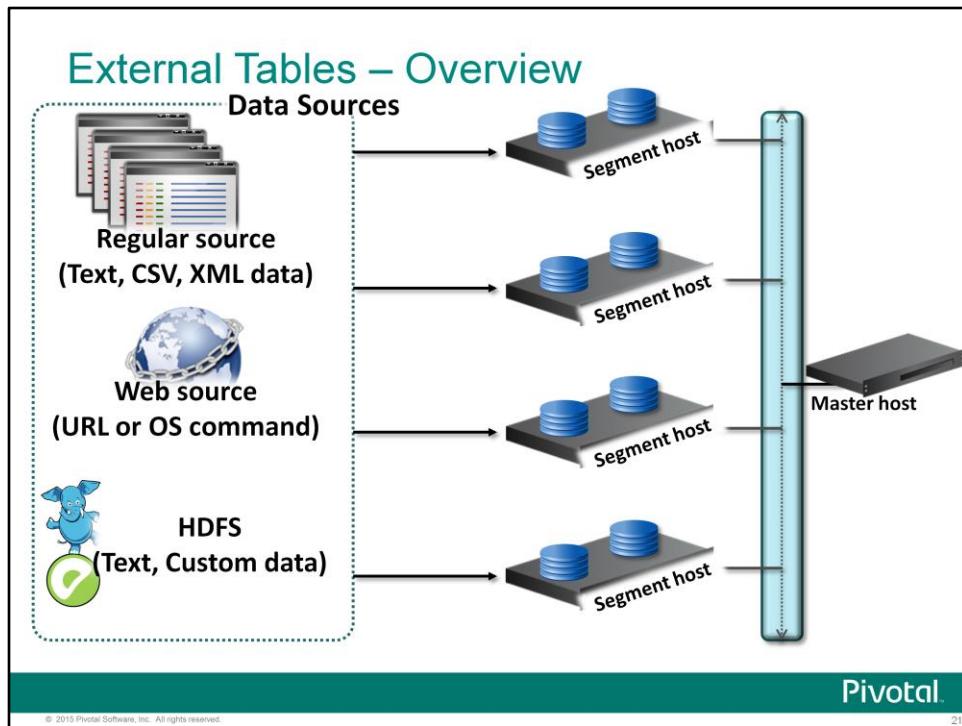
Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

20

You can define the following types of constraints:

- A column can be assigned a **DEFAULT** value.
- **CHECK** constraint is the most generic constraint type. It allows you to specify that the value in a certain column must satisfy a Boolean expression.
- **NOT NULL** constraint specifies that a column must not assume the null value.
- **UNIQUE** constraints ensure that the data contained in a column or a group of columns is unique with respect to all the rows in the table.



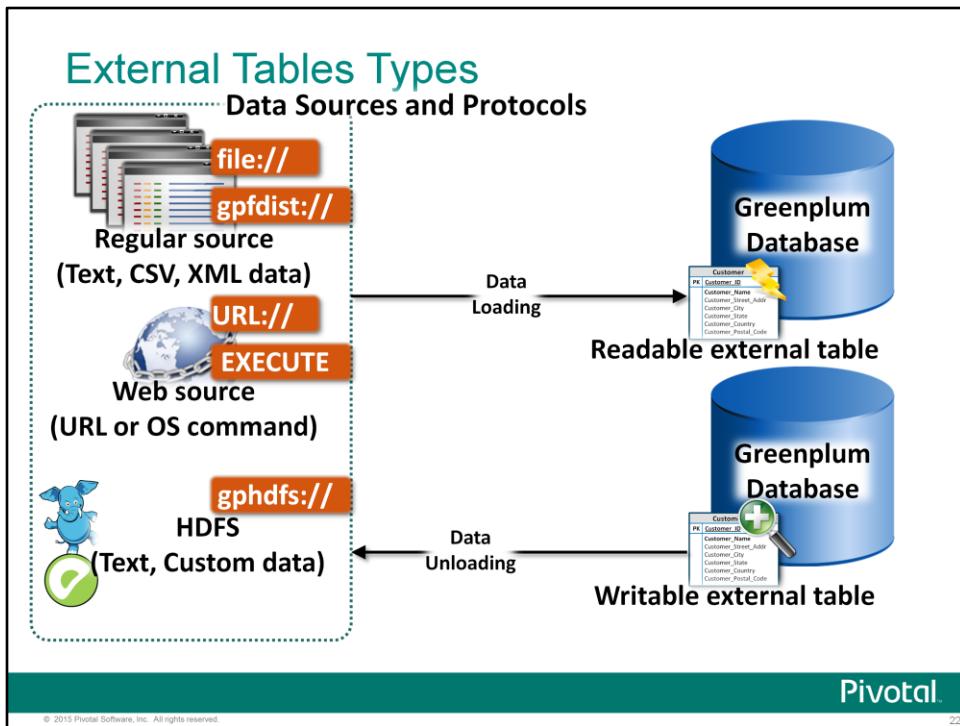
An external table is a definition of a table that specifies how to access a data source outside of the database as well as how that data is formatted. The format varies depending on the type of external table defined.

Large amounts of data can be loaded or unloaded using external tables. Greenplum supports fast, parallel data loading and unloading as well as non-parallel import and export of smaller amounts of data through one of several named protocols.

External tables lets you access data from a data source as though the data source was a regular table.

There are three main types of data sources that Greenplum works with:

- Regular or file-based external sources with support for text data, comma-separated values (CSV) formatted data, or XML data.
- Web-based external sources with support for text and CSV formatted data, as well as operating system commands and scripts.
- Hadoop file-system data sources or tables, with support for text-based data and custom or user-defined formatted data.



Greenplum supports two types of external tables:

- Readable or read-only tables which are used for data loading and cannot be modified.
- Writable or write-only tables which are used for data unloading.

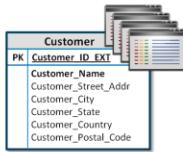
The protocol you use depends on the data source you are interfacing with:

- Regular or flat-file data sources are accessed using the file protocol, `file://`. The Greenplum parallel file distribution program, `gpfdist`, is referenced with the protocol, `gpfdist://`.
- External web tables access data sources with the HTTP protocol, `http://` or by accessing a command or script with `EXECUTE`. The command or script must exist on all segment servers if that is the method you are using.

Once an external table is defined, its data can be queried directly, and in parallel, using SQL commands. For example, you can issue `SELECT`, `JOIN`, or a `SORT` on external table data. You cannot modify the data on a readable external table.

A writable external table allows you to select rows from other database tables and output the rows to files, named pipes, or other executable programs. For example, you could unload data from Greenplum Database and send it to an executable that connects to another database or ETL tool to load the data elsewhere. Writable external tables can also be used as output targets for Greenplum parallel MapReduce calculations. Unlike a readable external table, `INSERT` operations are the only allowed operations. You cannot read from, update, delete, or truncate a writable external table.

# Regular, Web, and HDFS External Tables Summary



## Regular external tables:

- Access static flat files
- Allows for rescanning



## Web external tables:

- Access dynamic data sources
- Is not rescannable when a query is planned



## HDFS external tables:

- Require one-time setup
- Require read and/or write privileges to gphdfs protocol

## All external tables:

- Support query execution on external data
- Supports single-row error isolation to filter badly formatted rows
- Supports ETL loading and data unloads
- Is accessed by segments at runtime
- Should not be used for frequently executed queries

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

23

The main difference between regular external tables and web tables are their data sources:

- Regular tables access static flat files. The table is rescannable when a query is planned against the external table.
- Web tables access dynamic data sources, either on a web server or by executing OS commands or scripts. The data is not rescannable when a query is planned on the external table. This is because there is a possibility that the data will change during the course of the query's execution.
- Hadoop file system external tables lets you read and write data using the gphdfs protocol to and from Hadoop. As with other external tables, you can either read from or write to a data source. All data is read in parallel from the HDFS data nodes into the Greenplum segment servers for rapid processing of unstructured data. Each Greenplum segment instance reads one set of Hadoop data blocks and writes only the data contained within that segment to the Hadoop file system. The HDFS protocol requires an initial setup process, where Java is installed on all segment servers, Greenplum HD connectors and libraries are configured, gpadmin has read and write access to Hadoop libraries, and environmental variables and server configuration parameters are set. In addition to the initial configuration, any database users who will be using the HDFS protocol must be granted `SELECT` and/or `INSERT` privileges on the protocol if reading from or writing to data sources respectively.

## Regular, Web, and HDFS External Tables Summary (continued)

An external table:

- Can be thought of as a view that allows you to run SQL query against external data without first loading the data into the Greenplum database. External tables provide an easy way to perform basic extraction, transformation, and loading (ETL) tasks that are common in data warehousing. External table files are read in parallel by the Greenplum Database segment instances, so they also provide a means for fast data loading.
- Can be defined in *single-row error isolation mode* allowing you to filter out badly formatted rows while still loading good rows.
- Is accessed by segments at runtime in parallel. The same mechanisms used for parallel query execution forms the data into tuples and processes them as a regular parallel query would.
- Should not be used for frequently queried tables. As of now, there is no way to gather detailed statistics for external tables, so the query planner is not able to plan complex queries on external tables effectively.

## External Table SQL Commands

To manage external tables, use the following SQL syntax:

Action	SQL Syntax
Create a read-only external table	<code>CREATE EXTERNAL [WEB] TABLE LOCATION (...)   EXECUTE '...' FORMAT '...' (DELIMITER, '...');</code>
Create a writable external table	<code>CREATE WRITABLE EXTERNAL [WEB] TABLE LOCATION (...)   EXECUTE '...' FORMAT '...' (DELIMITER, '...') DISTRIBUTED BY(...)   RANDOMLY;</code>
Drop an external table	<code>DROP EXTERNAL [WEB] TABLE</code>

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

25

To create or remove an external table, use the Greenplum SQL syntax provided. When defining an external table:

- Table is given a name and columns are specified like a normal table definition. Unlike regular tables, external tables do not have column constraints or default values, so do not specify those.
- Use WEB to define the table as a web table. This is valid in both a read-only and write-only definition.
- The LOCATION clause specifies the location and type of the external data and the protocol used to access the external data (file, gpfdist, http, EXECUTE *os\_command*, gphdfs). If using the gpfdist:// protocol, you must have the Greenplum file distribution program, gpfdist, running on the host where the external data files reside. This program points to a given directory on the file host.
- The SEGMENT REJECT LIMIT clause is used to specify criteria for single row error handling. Normally a select from an external table is *all or nothing*, meaning it fails on the first format error encountered. This feature allows you to load good rows and ignore, or optionally log to an error table, badly formatted rows. Note this only applies to formatting errors, not constraint errors.
- FORMAT tells how the data is formatted, such as TEXT or CSV.

`DROP EXTERNAL TABLE` removes the table definition only – the data source remains intact and external files are not deleted.

## Table Tips

Here are some tips for obtaining table information in PSQL:

- To list tables in the database:  
  \dt
- To see structure of a table and distribution key columns:  
  \dt+ table\_name
- To list system catalog tables:  
  \dtS
- To list external tables only:  
  \dx
- To see data distribution or the number of rows on each segment:  
  SELECT gp\_segment\_id, count(\*)  
  FROM table\_name GROUP BY gp\_segment\_id;

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

26

Here are some helpful tips for working with tables in the PSQL client program.

## Data Types – Overview

Data types:

- Describe the type of data a column can contain
- In Greenplum are the same as data types in PostgreSQL
- Supported are defined in the SQL standard
- Can be created with the CREATE TYPE SQL command

For Greenplum distribution key columns:

- No geometric data types are supported
- No user-defined data types are supported

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

27

Data types describe the type of data a table column can contain.

Greenplum supports all data types defined in the SQL standard, and all the built in data types supported in PostgreSQL. If desired, you can also define your own custom data types.

For columns that are part of the Greenplum table distribution key, they cannot be of geometric or user-defined data types.

## Greenplum Database Data Types

Name	Size	Description
bigint	8 bytes	Large range integer
bigserial	8 bytes	Large autoincrementing integer
bit [(n)]	$n$ bits	Fixed-length bit string
bit varying [(n)]	Actual # of bits	Variable-length bit string
boolean	1 byte	Logical boolean (true/false)
box	32 bytes	Rectangular box in the plane (not allowed in distribution key columns)
bytea	1 byte + binary	Variable-length binary string
character [(n)]	1 byte + $n$	Fixed-length, blank padded
character varying [(n)]	1 byte + string size	Variable-length with limit
cidr	12 or 24 bytes	IPV4 and IPV6 networks
circle	24 bytes	Circle in the plane (not allowed in distribution key columns)

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

28

The table on this and the next two slides are for reference only. They describe the built-in data types available in Greenplum.

Each data type has an external representation determined by its input and output functions. Many of the built-in types have obvious external formats. However, several types are either unique to PostgreSQL and Greenplum Database, such as geometric paths, or have several possibilities for formats, such as the date and time types. Some of the input and output functions are not invertible. That is, the result of an output function may lose accuracy when compared to the original input.

For more details on the data types, including valid aliases and their defined range, refer to the *Greenplum Database Administrator Guide*.

## Greenplum Database Data Types (Cont)

Name	Size	Description
date	4 bytes	Calendar date (year, month, day)
decimal [(p,s)]	variable	User-specified precision, exact
double precision	8 bytes	Variable-precision, inexact
inet	12 or 24 bytes	IPv4 and IPv6 hosts and networks
integer	4 bytes	Usual choice for integer
interval [(p)]	12 bytes	Time span
lseg	32 bytes	Line segment in the plane <i>(not allowed in distribution key columns)</i>
macaddr	6 bytes	MAC addresses
money	4 bytes	Currency amount
path	16+16n bytes	Geometric path in the plane <i>(not allowed in distribution key columns)</i>
point	16 bytes	Geometric point in the plane <i>(not allowed in distribution key columns)</i>

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

29

## Greenplum Database Data Types (Continued)

## Greenplum Database Data Types (Cont)

Name	Size	Description
polygon	40+16n bytes	Geometric path in the plane <i>(not allowed in distribution key columns)</i>
real	4 bytes	Variable-precision, inexact
serial	4 bytes	Autoincrementing integer
smallint	2 bytes	Small range integer
text	1 byte + <i>string size</i>	Variable unlimited length
time [(p)] [without time zone]	8 bytes	Time of day only
time [(p)] [with time zone]	12 bytes	Time of day only, with time zone
timestamp [(p)] [without timezone]	8 bytes	Both date and time
timestamp [(p)] [with timezone]	8 bytes	Both date and time, with time zone

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

30

## Greenplum Database Data Types (Continued)

## Array Constructors – Overview

There are two types of arrays:

- An array constructor is an array of values from member elements

```
SELECT ARRAY[1,2,3+4];
array
-----
{1,2,7}
(1 row)
```

- A multidimensional or nested array constructor contains nested arrays

```
SELECT ARRAY[ARRAY[1,2],
ARRAY[3,4]];
array
-----
{{1,2},{3,4}}
(1 row)
```

OR

```
SELECT ARRAY[[1,2],[3,4]];
array
-----
{{1,2},{3,4}}
(1 row)
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

31

An array constructor is an expression that builds an array value from values for its member elements.

A simple array constructor consists of:

- The key word ARRAY
- A left square bracket ( [ )
- One or more comma separated expressions for the array element values
- A right square bracket ( ] )

For the example shown, there are three items in the array – the last item is a sum of two numbers.

Multidimensional arrays can be built by nesting array constructors. In the inner constructors, the key word ARRAY can be omitted.

In the examples shown at the bottom of the slide, the syntax shown with two ARRAY definitions is the same as the syntax shown with one ARRAY definition and nested brackets.

## Constants

There are three types of constants:

- Strings
- Bit strings
- Numbers

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

32

There are three kinds of implicitly-typed constants or literals in PostgreSQL:

- Strings
- Bit strings
- Numbers

Constants can be specified with explicit types which enable more accurate representation and more efficient handling of the constant by the system.

## Constants – Strings

A string:

- Is an arbitrary sequence of characters bounded by single quotes

EXAMPLE: 'This is a string.'

- Uses two single quotes to print a single quote

```
SELECT 'Dianne''s Horse';  
Dianne's Horse
```

- Can contain escape string constants. For example:

```
SELECT E'foo\'\tbar';  
foo' bar
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

33

A string constant in SQL is an arbitrary sequence of characters bounded by single quotes ('), for example 'This is a string'.

To include a single-quote character within a string constant, write two adjacent single quotes, for example, 'Dianne"’s horse'.

PostgreSQL also accepts *escape* string constants, which are an extension to the SQL standard.

## Dollar-quoted Strings

Dollar-quoted strings:

- Can make a query easier to read
- Uses dollar quoting to wrap a string
- Uses the following syntax:  
  \$ TAG\$String of words\$ TAG\$

A string can be represented in either of the following ways:

\$\$Dianne's horse\$\$

\$SomeTag\$Dianne's horse\$SomeTag\$

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

34

The standard syntax for specifying string constants is usually convenient, but it can be difficult to read, because of doubled quotes or backslashes.

To allow for more readable queries in such situations, PostgreSQL provides another way to write string constants, called *dollar quoting*.

A dollar-quoted string constant consists of:

- A dollar sign (\$)
- An optional “tag” of zero or more characters
- Another dollar sign
- An arbitrary sequence of characters that makes up the string content
- A dollar sign
- The same tag that began this dollar quote
- A dollar sign

## Constants – Escape Strings

Backslash characters are used to represent the following:

Escape Sequence	Description
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Tab
\digits	<i>digits</i> represent an octal byte value
\hexdigits	<i>hexdigits</i> represent a hexadecimal byte value

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

35

Within an escape string, a backslash character (\) begins a C-like backslash escape sequence, in which the combination of backslash and following characters represent a special byte value:

- \b is a backspace
- \f is a form feed
- \n is a newline
- \r is a carriage return
- \t is a tab

Also supported are:

- \digits, where digits represents an octal byte value
- \hexdigits, where hexdigits represents a hexadecimal byte value.

## Constants – Bit Strings

Bit strings:

- Use the format, `B'bit_value'` to represent a bit. For example:  
`B'1001'`
- Allows only 0's and 1's
- Can be defined in hexadecimal notation using the format, `X'hexadecimal_value'`. For example:  
`X'1FF'`
- Can be continued across lines
- Do not support dollar quoting

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

36

Bit-string constants look like regular string constants with a B (upper or lower case) immediately before the opening quote (no intervening whitespace), e.g., `B'1001'`.

The only characters allowed within bit-string constants are 0 and 1.

Alternatively, bit-string constants can be defined in hexadecimal notation, using a leading X (upper or lower case), e.g., `X'1FF'`.

This notation is equivalent to a bit-string constant with four binary digits for each hexadecimal digit.

Both forms of bit-string constant can be continued across lines in the same way as regular string constants.

Dollar quoting cannot be used in a bit-string constant.

## Constants – Numbers

Numeric constants are accepted as:

- *Digits*
- *digits.[digits][e[+‐]digits]*
- *[digits].digits[e[+‐]digits]*
- *digitse[+‐]digits*
- Examples of valid number constants:

42	3.5
4.	.001
5e2	1.925e-3

A numeric constant without a decimal point or exponent of size *n*:

- Is an integer if  $n \leq 32$  bits
- Is a bigint if  $32 > n \leq 64$  bits
- Is numeric if  $n > 64$

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

37

Numeric constants are accepted in these general forms:

- digits
- digits.[digits][e[+‐]digits]
- [digits].digits[e[+‐]digits]
- digitse[+‐]digits

where digits is one or more decimal digits, 0 through 9.

The following requirements must be met:

- At least one digit must be before or after the decimal point, if one is used.
- At least one digit must follow the exponent marker (e), if one is present.
- There cannot be any spaces or other characters embedded in the constant.

A numeric constant that contains neither a decimal point nor an exponent:

- Is presumed to be type `integer` if its value fits in the integer size, 32 bits.
- Is presumed to be type `bigint` if its value fits in bigint size, 64 bits.
- Is assumed to be numeric type if none of the above holds true

Constants that contain decimal points and/or exponents are always initially presumed to be type numeric.

## Casting Data Types

A type cast:

- Is used to convert one data type to another
- Can be issued with the following syntax:

Syntax	Example
type 'string'	REAL '2.117902'
'string'::type	'2.117902'::REAL
CAST ('string' AS type)	CAST ('2.117902' AS REAL)

- Can be omitted if there is no ambiguity
- Accepts regular SQL notation or dollar quoting for string constants

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

38

A type cast specifies a conversion from one data type to another.

A constant of an arbitrary type can be entered using any one of the following explicit notations:

- type 'string'                   REAL '2.117902'
- 'string' ::type                 '2.117902'::REAL
- CAST ('string' AS type)    CAST('2.117902' AS REAL)

The result is a constant of the indicated type.

The CAST syntax conforms to SQL; the syntax with :: is historical PostgreSQL usage.

If there is no ambiguity the explicit type cast can be omitted (for example, when constant it is assigned directly to a table column), in which case it is automatically converted to the column type.

The string constant can be written using either regular SQL notation or dollar-quoting.

## Additional Database Objects

Let us examine:

- Views
- Indexes
- Sequences
- Triggers
- Tablespaces

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

39

Now we will briefly cover the other database objects we haven't covered yet.

## Views – Overview

Views:

- Let you save frequently used or complex queries
- Can be accessed with `SELECT` statement
- Are not materialized on disk
- Are managed and accessed with the following commands:

Action	Command
Create a view	<code>CREATE VIEW</code>
Drop or remove a view	<code>DROP VIEW</code>
List all views	<code>\dv</code>
See view definition	<code>\dv+ view_name</code>

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

40

Views are a way to save frequently used or complex queries and then access them in a `SELECT` statement as if they were a table. A view is not physically materialized on disk as is a table. The query is instead ran whenever the view is accessed.

Note that currently Views ignore `ORDER BY` or `SORT` operations stored in the view, so run those in the query that calls the view.

Working with views in Greenplum Database is the same as in PostgreSQL or as you would expect in any other DBMS.

You can access and manage views with the following commands:

- `CREATE VIEW` – Creates a view
- `DROP VIEW` – Drops a view
- `\dv` – Meta-command to list all of the views in a database
- `\dv+ view_name` – Describes the view

At the bottom of the slide, you will find an example of a view being created.

## Views Example

The following example creates the view, `topten`:

```
Example: Creating the view, topten
CREATE VIEW topten
AS SELECT name, ranking, gender, year
FROM names, prod_rank
WHERE ranking < '11' AND names.id=prod_rank.id;
SELECT * FROM topten ORDER BY year, ranking;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

41

In this example, the view, `topten`, is created based on two tables: `names` and `prod_rank`. The data is restricted where the ID exists in both tables and ranking field is less than 11. After creating the view, you can issue a `SELECT` statement to view the rows extracted from the original table.

## Index – Overview

Indexes:

- Use random seek to find a record in a relational database
- Should be used sparingly in Greenplum Database
- Are not always favored at query runtime
- Should be checked to see if they improve performance
- Should be dropped if not used
- Are created automatically from a PRIMARY KEY
- Are allowed on Greenplum distribution key columns when columns are UNIQUE

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

42

Greenplum Database is very fast at sequential scanning. Unlike a traditional database, the data is distributed so that each segment scans a smaller portion of the overall data in order to get the result. Add in partitioning and the total data to scan gets even smaller. An index, which uses a random seek to find a record may not always improve query performance in Greenplum Database.

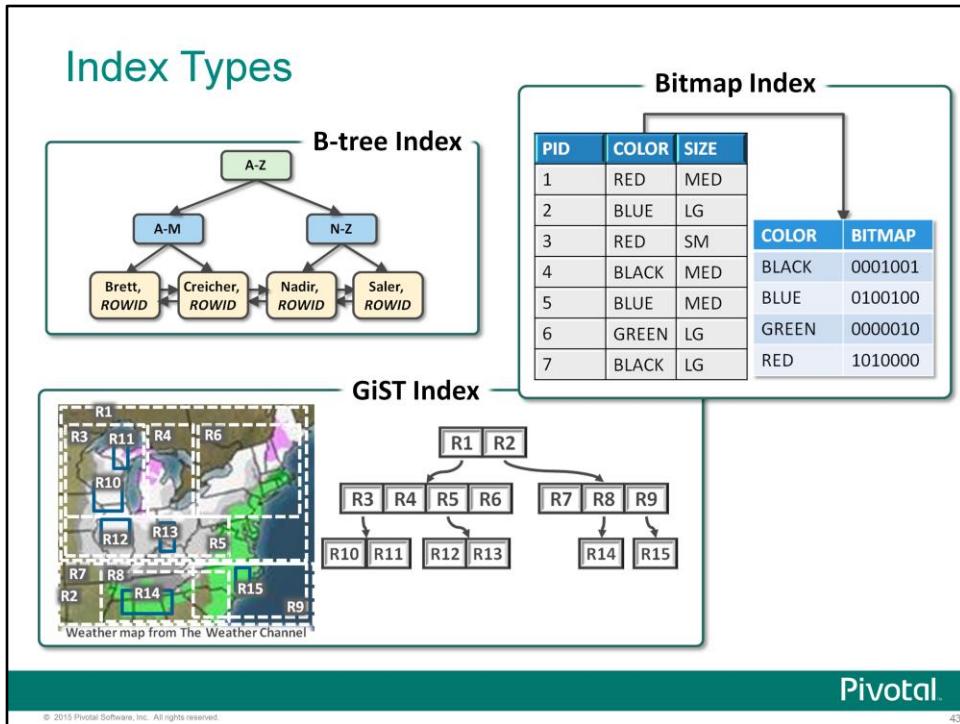
When deciding what indexes to create, examine the WHERE clauses of your query workload. On very large tables, these columns might still be good candidates for indexes.

Note that the query planner tends to favor a sequential scan over an index scan because a sequential scan is often determined by the planner to be faster. If the planner determines an index to be faster, it will use it.

An index does add some database overhead (the index must be maintained whenever a table is updated) so you want to ensure that the indexes you create are being used by your query workload and that they are improving query performance as compared to a sequential scan of the table.

Greenplum Database does have one limitation regarding UNIQUE indexes. To enforce the uniqueness of an index across the segments, unique indexes are only allowed on distribution key columns.

If you create a table with a PRIMARY KEY, a pkey index will be automatically created for that table.



Each index type uses a different algorithm that is best suited to different types of queries. Greenplum Database supports the following index types:

- B-tree (the default index type) fits the most common situations. B-trees can handle equality and range queries on data that can be sorted into some ordering.
- Bitmap is a new type of index only found in Greenplum and not in PostgreSQL.
- GiST indexes are not a single kind of index, but rather an infrastructure within which many different indexing strategies can be implemented.

## Commands to Manage Indexes

Commands to manage indexes include:

Action	Command	Greenplum Application
Create an index	CREATE INDEX	
Modify an index	ALTER INDEX	
Drop or remove an index	DROP INDEX	
Cluster an index	CLUSTER	
Re-index your indexes	REINDEX	reindexdb
List all indexes	\di	
View index definition	\d+ <i>index_name</i>	

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

44

You create, modify, or delete an index using the CREATE INDEX, ALTER INDEX, and DROP INDEX commands respectively.

CLUSTER physically reorders a table on disk based on the index information. This can be helpful for queries that access a range of data (by date for example – get June1 –June 30 2007 ). When an index is in order, the system does not have to hop around the disk seeking for records, but can access a range of records in sequence. Although the CLUSTER command is supported in Greenplum Database, it takes a long time to physically reorder large indexes across several segments. It is much faster to use CREATE TABLE ...AS SELECT ... ORDER BY <the index criteria> to physically reorder the table on disk in this situation.

The SQL command, REINDEX, and the Greenplum application, reindexdb are used to rebuild an index using the data stored in a table and replaces the old copy of an index. This should be used when:

- An index has become corrupted, and no longer contains valid data.
- An index has become bloated, that it is contains many empty or nearly empty pages. This can occur with B-tree indexes in Greenplum Database under certain uncommon access patterns.
- You have altered the fill factor storage parameter for an index and wish to ensure that the change has taken full effect.

Use the \di and \d+ *index\_name* meta-commands to list all of the indexes in a database and view details on the indexes respectively.

## Bitmap Indexes – Overview

- Bitmap indexes:
- Are ideal for data warehouse applications and decision support systems, where
  - There are large amounts of data and ad-hoc queries
  - There are low numbers of DML transactions
- Are a fraction of the storage size of B-tree indexes
- Are stored as a bitmap, not as tuple (row) IDs
- Uses a mapping function to convert bit positions to a tuple ID
- Are stored in a compressed way

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

45

Bitmap indexes are one of the most promising strategies for indexing high dimensional data in data warehousing applications and decision support systems. These types of applications typically have large amounts of data and ad hoc queries, but a low number of DML transactions.

Fully indexing a large table with a traditional B-tree index can be expensive in terms of space because the indexes can be several times larger than the data in the table. Bitmap indexes are typically only a fraction of the size of the indexed data in the table.

An index provides pointers to the rows in a table that contain a given key value. A regular index stores a list of tuple ids for each key corresponding to the rows with that key value. In a bitmap index, a bitmap for each key value replaces a list of tuple ids. Each bit in the bitmap corresponds to a possible tuple id, and if the bit is set, it means that the row with the corresponding tuple id contains the key value. A mapping function converts the bit position to an actual tuple id, so that the bitmap index provides the same functionality as a regular index. Bitmap indexes store the bitmaps in a compressed way. If the number of distinct key values is small, bitmap indexes compress better and the space saving benefit compared to a B-tree index becomes even better.

## When to Use Bitmap Indexes – Part 1

Bitmap indexes are best used for:

- Low-cardinality columns, where the number of distinct values to the number of rows is small

ROWID	LNAME	FNAME	MARITAL_STATUS
1	Worth	Sam	married
2	Priest	Lara	single
3	Jericho	Val	widowed
		...	
52000221	Mansk	Brett	divorced

Low cardinality

- Columns with less than 10,000 distinct values and less than 1% unique values
- Columns with less than 1% unique values

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

46

Bitmap indexes perform best for columns in which the ratio of the number of distinct values to the number of rows in the table is small. This ratio is known as the degree of cardinality. A marital status column, which has only a few distinct values (single, married, divorced, or widowed), is a good choice for a bitmap index, and is considered low cardinality.

However, columns with higher cardinalities can also have bitmap indexes. For example, on a table with one million rows, a column with 10,000 distinct values is also a good candidate for a bitmap index. A bitmap index on this column can outperform a B-tree index, particularly when this column is often queried in conjunction with other indexed columns. However, the performance gains start to diminish on columns with 10,000 or more unique values, regardless of the number of rows in the table.

A best practice rule to follow is that when less than 1% of the row count is distinct, use bitmap indexes.

## When to Use Bitmap Indexes – Part 2

Bitmap indexes are best used for:

- Tables that are not updated frequently
- Queries with multiple WHERE conditions

Query: Which of our customers  
who are female are also married?



Example: AND in the WHERE clause for two columns with bitmap index

```
SELECT * FROM customers WHERE gender = 'F' AND  
marital_status = 'married';
```

Column with  
bitmap index

Column with  
bitmap index

- Boolean operations are performed directly on the bitmaps
- Multiple WHERE clause conditions can be filtered first on the bitmap reducing table scans

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

47

Bitmap indexes can dramatically improve query performance for ad-hoc queries. AND and OR conditions in the WHERE clause of a query can be resolved quickly by performing the corresponding Boolean operations directly on the bitmaps before converting the resulting bitmap to tuple ids. If the resulting number of rows is small, the query can be answered quickly without resorting to a full table scan.

Bitmap indexes are most effective for queries that contain multiple conditions in the WHERE clause. Rows that satisfy some, but not all, conditions are filtered out before the table itself is accessed. This improves response time, often dramatically. FOR EXAMPLE - An analyst for the company might want to know, "Which of our customers who are female are also married?" This corresponds to the following SQL query:

```
SELECT * FROM customers WHERE gender = 'F' AND  
marital_status = 'married';
```

The bitmaps for gender='F' and marital\_status='married' are used to identify the rows that have the correct bit values, and the results are merged before accessing the actual tuple id in the customer table. This is more efficient than scanning the entire table multiple times, as only the exact rows that are needed are retrieved.

## When Not to Use Bitmap Indexes

Bitmap indexes are not good for:

- OLTP applications
- Frequently updated columns
- Unique or high-cardinality columns

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

48

Bitmap indexes should not be used for unique columns or columns with high cardinality data, such as customer names or phone numbers. B-tree indexes are a better choice for these types of columns.

Bitmap indexes are primarily intended for data warehousing applications where users query the data rather than update it. They are not suitable for OLTP applications with large numbers of concurrent transactions modifying the data.

## Bitmap Index – Example

user_id	gender=M	gender=F
1	1	0
2	0	1
3	1	0
4	1	0
5	1	0
6	0	1
7	1	0
8	0	1

Bitmap for **gender=M**: 10111010

Bitmap for **gender=F**: 01000101

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

49

Here is a simple table that illustrates a bitmap index for the gender column.

Each entry (or bit) in the bitmap corresponds to a single row of the table. The value of each bit depends upon the values of the corresponding row in the table. For example, the bitmap for gender='M' would look like this. It contains a '1' as its first bit because the gender is 'M' in the first row of the customer table. The bitmap index also keeps a list of values, which maps the bit to the actual column value. At runtime the value is mapped to a bit number, the bitmap is scanned, rows are identified by their position in the bitmap, and only the tuple ids that match the scan are retrieved.

## Sequences – Overview

A sequence:

- Is used to autoincrement unique ID columns
- Is generated by the Greenplum Master sequence generator process (seqserver)
- Has some function limitations:
  - `lastval` and `currval` not supported
  - `setval` cannot be used in queries that update data
  - `nextval` sometimes grabs a block of values for some queries. So values may sometimes be skipped in the sequence if all of the block turns out not to be needed at the segment level.
  - Sequences are based on bigint arithmetic



**Note:** bigserial and serial data types are auto incrementing integers.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

50

Creating a sequence generator involves creating and initializing a new special single-row table with the given sequence name. Sequences are often used to increment unique ID columns of a table whenever a new record is added.

In a regular, non-distributed, database, functions that operate on the sequence go to the local sequence table to get values as they are needed. In Greenplum, each segment is its own distinct PostgreSQL database. Therefore they need a single point of truth to go for sequence values so that all segments get incremented correctly and the sequence moves forward in the right order. A sequence server process runs on the master and is the point-of-truth for a sequence in Greenplum distributed database. Segments get sequence values at runtime from the master.

Because of this distributed sequence design, there are some limitations on the functions that operate on a sequence in Greenplum Database:

- `lastval` and `currval` are not supported.
- `setval` can only be used to set the value of the sequence generator on the master, it cannot be used in subqueries to update records on distributed table data.
- `nextval` will sometimes grab a block of values from the master for a segment to use, depending on the query. Values may sometimes be skipped in the sequence if all of the block turns out not to be needed at the segment level. Note that regular PostgreSQL does this as well, so this is not unique to Greenplum.

The bigserial and serial data types are auto incrementing integers. These can be used like sequences.

## Sequence SQL Commands

Manage and access sequences with the following commands:

Action	Command
Create a sequence	<code>CREATE SEQUENCE</code>
Modify a sequence	<code>ALTER SEQUENCE</code>
Drop or remove a sequence	<code>DROP SEQUENCE</code>
List all sequences	<code>\ds</code>
View sequence definition	<code>\d+ sequence_name</code>

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

51

Working with sequences in Greenplum is the same as in PostgreSQL or as you would expect in any other DBMS. Creating, altering, and dropping an index are managed with SQL commands.

Use the `\ds` and `\d+ sequence_name` meta-commands to list all of the sequences in a database and view the details on a sequence respectively.

## Sequence Example

Here are several examples of implementing sequences:

- Create a sequence named myseq

```
CREATE SEQUENCE myseq START 101;
```

- Insert a row into a table that gets the next value:

```
INSERT INTO distributors  
VALUES (nextval('myseq'), 'acme');
```

- Reset the sequence counter value on the master:

```
SELECT setval('myseq', 201);
```

- Not allowed in Greenplum Database (set sequence value on segments):

```
INSERT INTO product  
VALUES (setval('myseq', 201), 'gizmo');
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

52

The examples on the slide show how to:

- Create a sequence using the CREATE SEQUENCE syntax.
- Insert a row into the distributors table and call the next value of the sequence created. That value is inserted in the first column of the table.
- Reset the sequence counter to the value shown in the slide, 201.
- Reset the value of the sequence to 201 and insert that value into the product table.

## Triggers – Overview

Triggers:

- Triggers are considered to be a type of function
- The automatically execute a particular function when a specific type of operation is performed (like updates, deletes, and inserts)
- Have limited support in Greenplum, in that:
  - The function must be `IMMUTABLE` to execute on segment instances
    - The immutable function cannot execute any SQL or modify the database.
  - Triggers are not supported on Append-Optimized tables.
- Are fired in alphabetical order

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

53

Triggers are functions or procedural code that automatically executes whenever a specific type of operation is performed in the database. Triggers are often used for maintaining the integrity of data within the database.

Due to the distributed nature of a Greenplum Database system, the use of triggers is somewhat limited. The function used in the trigger must be `IMMUTABLE`, meaning it cannot use information not directly present in its argument list. The function specified in the trigger also cannot execute any SQL or modify the database in any way. Given that triggers are most often used to alter the database (for example, update these other rows when this row is updated), these limitations offer very little practical use of triggers in Greenplum Database.

If multiple triggers of the same kind are defined for the same event, they will be fired in alphabetical order by name.

Triggers are not supported on append-optimized tables.

## Trigger SQL Commands

To manage triggers, use the following SQL commands:

Action	SQL Syntax
Create a trigger	<code>CREATE TRIGGER name {BEFORE   AFTER} {event [OR ...]} ON table [ FOR [EACH] {ROW   STATEMENT} ] EXECUTE PROCEDURE funcname ( arguments )</code>
Drop a trigger	<code>DROP TRIGGER [IF EXISTS] name ON table [CASCADE   RESTRICT]</code>
Alter a trigger	<code>ALTER TRIGGER name ON table RENAME TO newname</code>

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

54

To create, drop, or modify a trigger, use the CREATE TRIGGER, DROP TRIGGER, and ALTER TRIGGER commands respectively.

## Tablespaces

Tablespaces:

- Provide an alternate location for tables and indexes
- Can be placed on faster drives
- Require superuser privileges to create or move data objects to it
- Require that a filespace is created for the directory location for each primary and mirror segment instance  
`gpfilespace -c gpfilespace_config_file`
- Are created using the following syntax:  
`CREATE TABLESPACE tablespace_name  
FILESPACE filespace_name`
- Are supported on systems that support symbolic links

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

55

A tablespace allows superusers to define an alternative location on the file system where the data files containing database objects, such as tables and indexes, may reside. This usage allows administrators to store specific tablespaces on faster drives, if the service level agreement requires it.

A user with appropriate privileges can pass the tablespace name to `CREATE DATABASE`, `CREATE TABLE`, or `CREATE INDEX` to have the data files for these objects stored within the specified tablespace.

In Greenplum Database, you must first create a filespace before creating a tablespace that is associated with the directories defined in the filespace. The filespace is the directory location for each segment instance (primary and mirror) in your Greenplum Database array. You use the `gpfilespace` command to first create the filespace. You can use the same location for all your primaries if you want and a separate location for your mirrors (if they are using the same set of hosts as your primaries). The directory must be created before creating the tablespace. Use the `gpfilespace` command to create that filespace.

Tablespaces are only supported on systems that support symbolic links.

Note that the Greenplum implementation of tablespaces is an extension of the SQL standard.

Once the tablespace has been created, data objects can be created in the table space.

## Lab: Data Definition Language

In this lab, you create and manage database objects using the Data Definition Language. This includes creating databases, schemas, tables, views, indexes, and schemas.

You will:

- Create a database called `faa` and a schema called `faadata`. You will set the schema search path and learn how to verify which schema you are in.
- Create some tables in Greenplum Database and learn how the Greenplum distribution key for a table is chosen.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

56

In this lab, you create and manage database objects using the Data Definition Language. This includes creating databases, schemas, tables, views, indexes, and schemas.

# Module 4: Defining and Securing the User Database

## Lesson 1: Summary

During this lesson the following topics were covered:

- Databases overview
- Schemas and tables
- Constraints
- Data types and constants
- Views, indexes, sequences, triggers, and tablespaces

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

57

This lesson covered an overview of databases, schemas, tables, and how to create and manage these objects. A description of constraints and the various types supported by Greenplum Database, as well as a list and description of various data types supported within the environment. Additionally, the lesson covered descriptions and syntax for creating and managing views, indexes, sequences, triggers, and tablespaces

## Module 4: Defining and Securing the User Database

### Lesson 2: Data Manipulation Language and Data Query Language

In this lesson, you examine data manipulation language and data query language syntax to access and manage data.

Upon completion of this lesson, you should be able to:

- Describe the SQL support in the Greenplum Database
- Describe concurrency control
- Use functions and operators to build queries
- Examine how transaction concurrency is handled in Greenplum Database

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

58

Data manipulation language consists of SQL syntax that is used to manage and change data. This includes inserting, deleting, and updating data in tables. Data query language is the art of extracting and retrieving data.

Upon completion of this lesson, you should be able to:

- Describe the SQL support available in the Greenplum Database
- Describe concurrency control
- Insert, update, and delete data using SQL syntax
- Use functions and operators to build queries
- Examine how transaction concurrency is handled in Greenplum Database using MVCC

## SQL Support in Greenplum Database

Greenplum Database:

- Supports:
  - The majority of features in the SQL 1992 and SQL 1999 standards
  - Several features in the SQL 2003 and SQL 2008 standards
- Intends to fully support all SQL commands as defined in PostgreSQL
- Has some limitations to DDL commands, including:
  - Triggers                    — External Tables
  - Foreign keys              — Workload Management
- Includes support for features not in PostgreSQL, including:
  - Parallelism
  - OLAP

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

59

Both data manipulation language (DML) and data query language (DQL) are supported as in PostgreSQL with some minor limitations. Greenplum supports:

- The majority of features of the SQL 1992 and SQL 1999 standards
- Several of the features of the SQL 2003 and SQL 2008 standards

All other SQL commands are considered Data Definition Language (DDL) or utility commands. Most DDL is supported as in PostgreSQL, but there are a few limitations regarding things like foreign keys and triggers.

Greenplum has also added SQL to support features not in regular PostgreSQL, including:

- Parallelism
- OLAP SQL extensions to SELECT
- Managing external tables
- Managing workload management with resource queues

For details on the supported features in the SQL standards, refer to the *Greenplum Database Administrator Guide*.

## Concurrency Control and Multi-version Concurrency Control Features

### Data consistency:

- Is maintained by using the MVCC model (Multi-version Concurrency Control).
- Lets each transaction see a snapshot of data
- Protects the user from viewing inconsistent data that could be caused by other transactions executing concurrent updates on the same data rows

### MVCC:

- Provides transaction isolation for each database session.
- Uses locking methodologies to minimize lock contention
- Ensures reading never blocks writing and writing never blocks reading

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

60

When you start a transaction, you snapshot the data at the time in point you are looking at it. The advantage of this is that all of the data is captured at that time. If a query takes 3 hours to complete, instead of working on the data that exists after the query completes, it works on the data at the time of the snapshot. This eliminates inconsistencies that can occur when data is changed during that time period, such as when:

- Inserting new rows
- Updating existing rows
- Deleting rows

This provides transaction isolation for each database session. By eschewing explicit locking methodologies of traditional database systems, multi-version concurrency control, or MVCC:

- Minimizes lock contention in order to allow for reasonable performance in multiuser environments.
- Ensures locks acquired for querying, or reading, data do not conflict with locks acquired for writing data. Reading never blocks writing and writing never blocks reading.

Greenplum Database provides various lock modes to control concurrent access to data in tables. Most Greenplum Database SQL commands automatically acquire locks of appropriate modes to ensure that referenced tables are not dropped or modified in incompatible ways while the command executes. For applications that cannot adapt easily to MVCC behavior, the `LOCK` command can be used to acquire explicit locks. However, proper use of MVCC will generally provide better performance than locks.

## Managing Data

When working with data, data can be:



Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

61

Data is managed using the following concepts:

- **Insert** – Data is inserted into tables
- **Update** – Existing data is updated or modified
- **Delete** – Existing data is removed from a table

## Inserting Data

INSERT command:

- Is fully supported
- Does not match the performance of COPY or external tables
- Can be substituted with:

```
INSERT INTO <table>
    SELECT FROM <external table>
or
COPY
```

The following is an example of its use:

```
INSERT INTO names VALUES
    (nextval('names_seq'), 'test', 'U');
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

62

While the INSERT command contains the same level of support as in PostgreSQL, the performance is not as good in the distributed environment. If you have a lot of rows to insert, consider using COPY or external tables for faster load performance.

## Updating Data

The UPDATE command:

- Is used to update individual, multiple, or all rows in a table
- Requires the same distribution key be used when joining using the distribution column and an equijoin
- Does not allow distribution key columns to be updated

The following is an example of its use:

```
UPDATE names SET name='Emily' WHERE name='Emmmily';
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

63

The UPDATE command can be used to update rows in Greenplum DB.

Updating data refers to the modification of data that already exists in the database. You can update individual, multiple, or all rows in a table.

Greenplum has the following restrictions or limitations:

- When joining two tables using the UPDATE command, the tables must have the same Greenplum distribution key and be joined on the distribution key column(s). This is referred to as an equijoin.
- Greenplum distribution key columns may not be updated.

With the MVCC transaction model, every update is essentially an upsert, where the updated row's visibility information is marked stale and a newly visible row is inserted.

## Deleting Data

The following commands are used to remove data:

- **DELETE** command has the limitation where any table joins must be equijoins

```
DELETE FROM ranking; (deletes all rows)  
DELETE FROM ranking WHERE year='2001';
```

- **TRUNCATE** command deletes rows in a specified table

```
TRUNCATE b2001;
```

- **DROP** command deletes all rows and the table definition

```
DROP TABLE ranking;
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

64

The following commands can be used to delete data in Greenplum DB:

- **DELETE** – Greenplum has a limitation when joining two tables in a **DELETE** statement. The tables to be joined must have the same distribution key columns if the join is being performed on the distribution column and be joined using those columns (equijoins).
- **TRUNCATE** – This is like an unqualified delete, but it is faster since it does not scan the tables as the **DELETE** command does. As **TRUNCATE** does not scan tables, it does not truncate child tables, working only on the table specified.
- **DROP** – This command drops the rows and the table definition specified.

For parallel deletes, the **WHERE** clause must be on the distribution columns. The same for is true for the **UPDATE** command.

## Selecting Data

The `SELECT` statement:

- Selects data from a table
- Supports the same set of standards as is supported with PostgreSQL

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

65

The support for `SELECT` statements in the Greenplum Database is the same as PostgreSQL. This now includes correlated subquery, which enjoyed less support in Greenplum Database in earlier revisions of the database.

## Correlated Subqueries

Greenplum can flatten correlated subqueries that:

- Is a nested `SELECT` statement
- Refers to a column from an outer `SELECT` statement
- Does not need input from parent query
- Produces a scalar value or table values

Example: Correlated subquery finding all flight numbers not associated with a specific tail number

```
SELECT * FROM transaction t WHERE t.salesamt > 50 AND t.storeid=
  (SELECT s.storeid FROM store s WHERE s.countrycd='CAN' AND s.storeid=t.storeid);
```

Subquery

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

A correlated subquery is a nested `SELECT` statement that refers to a column from an outer `SELECT` statement. A subquery can be scalar, in that it returns a single row with a single value, or a table subquery, where multiple rows can be returned.

Greenplum Database supports correlated subqueries by unnesting the subquery where possible, or transforming the subquery into a join with the tables of the outer query.

If a correlated subquery has characteristics that do not prevent flattening, then it will be transformed to a join and be executed as expected.

It is important to understand the potential performance impacts on a query. A correlated query may require a rewrite in order to improve the overall performance of the query.

In this specific case, the cost of the correlated subquery is greater than a rewrite. A rewrite on the query displayed on the screen is:

```
SELECT t.* FROM transaction t, store s
WHERE t.salesamt > 50
AND s.countrycd='CAN'
AND s.storeid=t.storeid;
```

## Common Table Expressions

A common table expression:

- Is a temporary table used in SELECT statements
- Does not require permissions changes to the schema
- Is considered a transient view
- Is also known as the `WITH` clause

The screenshot shows a slide from Pivotal Software. At the top, there's a title bar with a magnifying glass icon and the text "Example: Obtain a random sampling of 10 unique airline IDs and get a report on the flight numbers associated with those airline IDs using a common table expression". Below this is a code block with a callout pointing to the first line: "WITH random\_sampler\_cte AS". The callout is labeled "CTE definition". The code itself is as follows:

```
WITH random_sampler_cte AS
  (SELECT DISTINCT(airlineid) FROM factontimeperformance
   LIMIT 10)
  SELECT flightnum, origincityname, originstatename
    FROM factontimeperformance, random_sampler_cte
   WHERE random_sampler_cte.airlineid=
         factontimeperformance.airlineid;
```

At the bottom right of the slide is the Pivotal logo.

The common table expression, introduced in SQL standard 2003, is a temporary table set used in subsequent SELECT statements. It is often preferred over views, in that views require permissions to the schema, necessitating privilege changes. Common table expressions however are transient views, derived from a query. It is often called the *WITH clause*.

CTEs address queries that are generated by business intelligence tools. These are not used as often in adhoc queries because of their potential complexities.

## Built-in Functions and Operators

Greenplum:

- Supports all categories of built-in functions and operators provided by PostgreSQL
- Provides full support of `IMMUTABLE` functions
- Provides limited use of `STABLE` and `VOLATILE` functions

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

68

PostgreSQL provides a large number of functions and operators for the built-in data types. Greenplum fully supports all built-in functions that are defined as immutable. An immutable function always returns the same result when given the same argument values; that is, it does not do database lookups or otherwise use information not directly present in its argument list.

For functions that are stable or volatile, they can be used in a limited capacity in Greenplum Database.

The function types are defined as follows:

- **STABLE** indicates that within a single table scan the function will consistently return the same result for the same argument values, but that its result could change across SQL statements. Functions whose results depend on database lookups or parameter variables are classified as `STABLE`. Also note that the `current_timestamp` family of functions qualify as stable, since their values do not change within a transaction.
- **VOLATILE** indicates that the function value can change even within a single table scan. Relatively few database functions are volatile in this sense; some examples are `random()`, `currval()`, `timeofday()`. But note that any function that has side-effects must be classified volatile, even if its result is quite predictable (for example, `setval()`).

## Stable and Volatile Functions

Stable and volatile functions:

- Can be used in statements that are evaluated on the master (no FROM clause):

```
SELECT setval('myseq', 201);  
SELECT foo();
```
- Cannot be used in statements that execute at the segment level if the function contains SQL or modifies the database in any way:

```
SELECT * FROM foo();
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

69

In Greenplum Database, the data is divided up amongst the segments. Each segment is, in a sense, its own distinct PostgreSQL database. To prevent data from becoming out-of-sync across the segments, any function classified as STABLE or VOLATILE cannot be executed at the segment level if it contains SQL or modifies the database in any way. For example, functions such as `random()` or `timeofday()` are not allowed to execute on distributed data in Greenplum Database because they could potentially cause inconsistent data between the segment instances. Anything in a `FROM` or `WHERE` clause is evaluated at the segment level. The following statement executes on the segments (has a `FROM` clause), however it may be allowed provided that the function used in the `FROM` clause simply returns a set of rows.

To ensure data consistency, VOLATILE and STABLE functions can safely be used in statements that are evaluated on and execute from the master. For example, statements without a `FROM` clause are always evaluated at and dispatched from the master.

## Built-in Functions (SELECT)

Function	Description	Example
CURRENT_DATE	Returns the current system date	2006-11-06
CURRENT_TIME	Returns the current system time	16:50:54
CURRENT_TIMESTAMP	Returns the current system date and time	2008-01-06 16:51:44.430000+00:00
LOCALTIME	Returns the current system time with time zone adjustment	19:50:54
LOCALTIMESTAMP	Returns the current system date and time with time zone adjustment	2008-01-06 19:51:44.430000+00:00
CURRENT_ROLE ROLE	Returns the current database user	jdoe

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

70

The functions shown are built-in functions that can be used as part of a SELECT statement.

## Mathematical Functions

Function	Returns	Description	Example	Results
+ - * /	same	Add, Subtract, Multiply & Divide	1 + 1	2
%	Integer	Modulo	10%2	0
^	Same	Exponentiation	2^2	4
/	Numeric	Square Root	/9	3
/	Numeric	Cube Root	/8	2
!	Numeric	Factorial	!3	6
&   # ~	Numeric	Bitwise And, Or, XOR, Not	9 & 15	11
<< >>	Numeric	Bitwise Shift left, right	1 << 4 8 >> 2	16 2

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

71

Commonly used mathematical functions shown on this and the next slide are supported in Greenplum Database and PostgreSQL. These tables are provided as reference only. For a detailed listing of the functions supported, refer to the *Greenplum Database Administrator Guide*.

## Mathematical Functions (Cont)

Function	Returns	Description	Example	Results
abs	same	Absolute Value	abs(-998.2)	998.2
ceiling (numeric)	Numeric	Returns smallest integer not less than argument	ceiling(48.2)	49
floor (numeric)	Numeric	Returns largest integer not greater than argument	floor(48.2)	48
pi()	Numeric	The $\pi$ constant	pi()	3.1419...
random()	Numeric	Random value between 0.0 and 1.0	random()	.87663
round()	Numeric	Round to nearest integer	round(22.7)	23

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

72

## Mathematical Functions (Continued)

## String Functions

Function	Returns	Description	Example	Results
string    string	Text	String concatenation	'my'    'my'	'mymy'
char_length(string)	Integer	number of chars in string	char_length('mymy')	4
position(string in string)	Integer	Location of specified sub-string	position('my' in 'ohmy')	3
lower(string)	Text	Converts to lower case	lower('MYMY')	'mymy'
upper(string)	Text	Converts to upper case	upper('mymy')	'MYMY'
substring(string from n for n)	Text	Displays portion of string	substring('myohmy' from 3 for 2)	'oh'
trim(both,leading,trailing from string)	Text	Remove leading and/or trailing characters	trim(' mymy ')	'mymy'

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

73

These commonly used string functions are supported in Greenplum Database and PostgreSQL. For a complete listing of the string functions available, refer to the *Greenplum Database Administrator Guide*.

## String Functions (Cont)

Function	Returns	Description	Example	Results
initcap (string)	Text	Changes case	initcap ('my my')	'My My'
length (string)	Integer	Returns string length	length ('mymy')	4
split_part (string, delimiter, occurrence)	Text	Separates delimited list	split_part('one two three',' ',2)	'two'

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

74

## String Functions (Continued)

Date Functions				
Function	Returns	Description	Example	Results
age(timestamp,timestamp)	Timestamp	Difference in years, months and days	age('2008-08-12' timestamp, current_timestamp)	0 years 1 month 11 days
extract (field from timestamp)	Integer	Returns year, month, day, hour, minute or second	extract( day from current_date)	11
now()	Timestamp	Returns current date & time	now()	2008-09-22 11:00:01
overlaps	Boolean	Simplifies comparing date ranges	WHERE ('2008-01-01','2008-02-11') overlaps ('2008-02-01','2008-09-11')	TRUE

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

75

Commonly used date functions are displayed on the slide. For detailed information and a full listing of all date functions, refer to the *Greenplum Database Administrator Guide*.

## Date Manipulation Examples

Here is an example to get the first day of the month:



Example: Retrieve the first day of the current month

```
SELECT CURRENT_DATE -  
       EXTRACT( DAY FROM CURRENT_DATE )::int + 1;
```

*You can use this for ANY DATE column in the data warehouse! It can save you time on coding joins to the CALENDAR tables.*



Example: Retrieve the first day of any month from facts.transaction

```
SELECT  
       ( t.transdate -  
         EXTRACT(DAY FROM t.transdate)::int + 1) AS  
         MonthStartDate, t.*  
FROM  
       facts.transaction t;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

76

In the examples shown, you can retrieve the first day of the month using the examples shown on the slide. The CURRENT\_DATE and EXTRACT functions are used to derive that data in the first example.

The second example allows you to retrieve the first day of any month provided in the data.

## Statistical or Aggregate Functions

Function	Returns	Description
sum	bigint for smallint or int arguments, numeric for bigint arguments, double precision for floating-point arguments, otherwise the same as the argument data type	Sum of <i>expression</i> across all input values
count	bigint	Number of input rows for which the value of <i>expression</i> is not null
avg	numeric for any integer type argument, double precision for a floating-point argument, otherwise the same as the argument data type	the average (arithmetic mean) of all input values
min	same as argument type	Minimum value of <i>expression</i> across all input values
max	same as argument type	Maximum value of <i>expression</i> across all input values

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

77

An aggregate function is a function that acts on values within a column. The aggregate function computes multiple input values to a single output value. For example, calculating the total of values in a column can be performed with the SUM aggregate function.

The table shows some of the more commonly used aggregates: SUM, COUNT, AVG, MIN, and MAX. Aggregates are not allowed in the WHERE clause, but are allowed in the HAVING clause.

## Statistical Functions – SUM

SUM:

- Returns the arithmetic sum of all values from a specified column:



### Example: Sum of all values in itemcnt

```
SELECT    SUM(itemcnt) as ItemQtySold  
FROM      facts.transaction;
```

- When used with a GROUP BY can determine the sum of column values grouped by another column:



### Example: GROUP BY column\_name

```
SELECT    SUM(itemcnt), storeid  
FROM      facts.transaction  
GROUP BY storeid;
```



### Example: GROUP BY position

```
SELECT    SUM(itemcnt), storeid  
FROM      facts.transaction  
GROUP BY 2;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

78

The SUM function returns the arithmetic function of all values from a specified column. It can also be used with the GROUP BY expression to determine the sum of column values grouped by another column.

## Statistical Functions - COUNT

COUNT:

- Returns the number of qualified rows in a result tab.

```
SELECT COUNT(*)  
FROM facts.transaction;
```

- Can be used to count distinct values with the DISTINCT operator

```
SELECT COUNT(DISTINCT storeid)  
FROM facts.transaction;
```

- Can be used with GROUP BY to determine the number of rows grouped by a selected column

```
SELECT storeid, COUNT(*)  
FROM facts.transaction  
GROUP BY 1;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

79

The COUNT function returns the number of qualified rows in a result tab. You can obtain the number of unique values in a column by adding the DISTINCT operator to the COUNT function. The example shown returns a count of the number of unique storeids in the transaction table.

The COUNT function used with a GROUP BY can be useful for determining the number of rows grouped by a selected column, as shown by the third example on the slide.

## Statistical Functions – AVG

AVG:

- Returns the arithmetic average of a specified column

```
SELECT      AVG(totalamt)
FROM        facts.transaction;
```

- Can be used with GROUP BY to determining the average of values, grouped by another column

```
SELECT      transdt,  AVG(totalamt)
FROM        facts.transaction
GROUP BY    1;
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

80

AVG returns the arithmetic average of a specified column in a result table. The query in the first example on the slide shows the average of the total sales amounts from the transaction table.

The AVG function, used with GROUP BY can be useful for determining the average amounts of values, grouped by another column. The second example on the slide obtains the daily average of the totalamt column from the transaction table.

## Statistical Functions – MIN and MAX

MIN returns the minimum value of a given column

```
SELECT  MIN(transdt)  
FROM    facts.transaction;
```

MAX returns the maximum value of a given column

```
SELECT  MAX(transdt)  
FROM    facts.transaction;
```



**Note:** Adding additional dimensions to the select statement requires a GROUP BY for each dimension. This returns the MIN or MAX (or both), aggregated over all dimensions.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

81

MIN returns the minimum value of a given column. The query in the first example shows the earliest transaction date value from the transaction table.

MAX returns the maximum value of a given column. The query in the second example on the slide returns the maximum transaction date value from the transaction table.

Note that adding additional dimensions to the select statement requires a GROUP BY for each dimension. This returns the MIN or MAX (or both), aggregated over all dimensions.

## Logical Operators

x	y	x AND y	x OR y
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE
TRUE	NULL	NULL	TRUE
FALSE	NULL	NULL	FALSE
NULL	NULL	NULL	NULL

x	NOT x
TRUE	FALSE
FALSE	TRUE
NULL	NULL

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

82

Logical operators AND and OR are displayed on the slide. These return a boolean (TRUE or FALSE) based on the operation performed.

## POSIX Pattern Matching

Operator	Description	Example
<code>~</code>	Matches regular expression Case sensitive	<code>'thomas' ~ '.*thomas.*'</code>
<code>~*</code>	Matches regular expression Case insensitive	<code>'thomas' ~* '.*Thomas.*'</code>
<code>!~</code>	Does not match regular expression Case sensitive	<code>'thomas' !~ '.*Thomas.*'</code>
<code>!~*</code>	Does not match regular expression Case insensitive	<code>'thomas' !~* '.*vadim.*'</code>

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

83

Pattern matching and the use of regular expression can greatly simplify your string manipulation and comparison needs.

## Comparison Operators

Operator	Description
=	Equal to
!= OR <>	NOT Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
x BETWEEN y AND z	Short hand for x >= y <u>and</u> x <= z
x IS NULL	True if x has NO VALUE
'abc' LIKE '%abcde%'	Pattern Matching
POSIX Comparisons	POSIX Pattern Matching

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

84

Comparison operators are used to compare two objects to each other.

## What Is NULL?

Nulls:

- Represent the absence of value
- Are a place holder indicating that no value is present
- Can be replaced based on business rules

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

85

`NULL` is the value used to represent an unknown piece of data in a column or field. It represents the absence of a value.

Within your data files you can designate a string to represent null values.

The default string is:

- `\N` (backslash-N) in TEXT mode
- An empty value with no quotations in CSV mode

You can also declare a different string using the `NULL` clause of `COPY`, `CREATE EXTERNAL TABLE` or `gpload` when defining your data format. It acts as a placeholder indicating that no value is present. For example, you might prefer an empty string for cases where you do not want to distinguish nulls from empty strings. When using the Greenplum Database loading tools, any data item that matches the designated null string will be considered a null value.

## User-defined Functions

Greenplum Database:

- Supports server extensibility as does PostgreSQL
- Limited use of `STABLE` and `VOLATILE` functions
- Lets you register new functions with the following requirements:
  - Declare appropriate volatility level (`VOLATILE`, `STABLE`, `IMMUTABLE`); if not declared, `VOLATILE` is assumed.
  - Place shared library files in the same library path location on every host (master, segments, mirrors)
  - Any functions used for user-defined data types, operators or aggregates must be `IMMUTABLE`

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

86

Both Greenplum Database and PostgreSQL support user-defined functions. The process for extending the server functionality with new functions, and thereby data types, aggregates and operators, is the same as in PostgreSQL.

After you write a function, you must register it with the server using `CREATE FUNCTION`. When doing this make sure to choose the correct volatility level and keep in mind the limitations of `STABLE` and `VOLATILE` functions we discussed on the previous slide. The default level is `VOLATILE`. Any function used in a user-created data type, aggregate, or operator must be `IMMUTABLE` in Greenplum Database or else you will get errors when trying to use the function.

Note that in Greenplum Database, the shared library files for C user-created functions must reside in the same library path location on every host in the Greenplum Database array (masters, segments, and mirrors). Best to place these files relative to the `$GPHOME/lib` directory.

## Transactions

Transactions:

- Bundle multiple statements into one *all-or-nothing* operation

Action	SQL Syntax
Start a transaction block	BEGIN or START TRANSACTION
Commit the results of a transaction	END or COMMIT
Abandon the transaction	ROLLBACK
Create a savepoint	SAVEPOINT

Autocommit mode:

- Is enabled by default in psql
- Can be turned off with \set autocommit on|off



**Note:** Two-phase commit transactions are not supported

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

87

Transactions allow you to bundle together multiple SQL statements in one all-or-nothing operation. They also ensure that the changes are indeed made to permanent storage won't be lost if a crash occurs.

The SQL commands used to perform transactions are:

- **BEGIN or START TRANSACTION** – Start a transaction block
- **END or COMMIT** – Commit the results of the transaction
- **ROLLBACK** – Abandon the transaction without changes.
- **SAVEPOINT** – Savepoints allow you to selectively discard parts of the transaction while committing the rest. After defining a savepoint with the **SAVEPOINT** command, you can roll back to a savepoint with the **ROLLBACK TO** command.

By default, psql runs in **autocommit** mode. This means each command is implicitly wrapped in a **BEGIN** and **COMMIT** (or a **ROLLBACK** if there is an error). Each statement issued in psql is its own transaction. You may want to run several commands in a single transaction, so it may be desired to sometimes turn autocommit off. With autocommit off, you will have to explicitly use **BEGIN** to start a transaction and **COMMIT** or **ROLLBACK** to end it.

Postgres 8.1 introduced the concept of 2 phase commit transactions. After a transaction is prepared, the transaction is no longer associated with the current session; instead, its state is fully stored on disk, and there is a very high probability that it can be committed successfully, even if a database crash occurs before the commit is requested. This feature is currently not supported in Greenplum DB.

## Transaction Concurrency Control

Greenplum supports all transaction isolation levels, including:

- READ COMMITTED/READ UNCOMMITTED
- SERIALIZABLE/REPEATABLE READ

In Greenplum:

- INSERT/COPY acquire locks at the row-level
- UPDATE/DELETE acquire locks at the table-level
- You can use the LOCK command to acquire specific locks
- The LOCK command must be used within a transaction block

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

88

The SQL standard defines four transaction isolation levels:

- **READ COMMITTED** – This is the default behavior for transaction isolation. A statement can only see rows committed before it began executing.
- **READ UNCOMMITTED** – In Greenplum Database, READ UNCOMMITTED is the same as READ COMMITTED.
- **SERIALIZABLE** – All statements of the current transaction can only see rows committed before the first statement was executed in the transaction. SERIALIZABLE is the strictest transaction isolation. This level emulates serial transaction execution, as if transactions had been executed one after another, serially, rather than concurrently. Applications using this level must be prepared to retry transactions due to serialization failures.
- **REPEATABLE READ** – In Greenplum Database, REPEATABLE READ is the same as SERIALIZABLE.

Row-level locks are acquired with the INSERT and COPY commands, while table-level locks are acquired using the UPDATE and DELETE commands.

If the MVCC model does not provide the level of concurrency protection you need, you can acquire explicit locks on a table using the LOCK command. Once a lock is acquired, it is held until the end of the transaction. The LOCK command must be used within a BEGIN/END transaction block to hold the lock on the object you specify.

## Lock Modes

Lock Mode	SQL Commands	Conflicts With
ACCESS SHARE	SELECT	ACCESS EXCLUSIVE
ROW SHARE	SELECT FOR UPDATE, SELECT FOR SHARE	EXCLUSIVE, ACCESS EXCLUSIVE
ROW EXCLUSIVE	INSERT, COPY	SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE
SHARE UPDATE EXCLUSIVE	VACUUM (without FULL), ANALYZE	SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE
SHARE	CREATE INDEX	ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

89

The table shows the available lock modes and the contexts in which they are used automatically by Greenplum Database, and the lock modes they conflict with.

You can acquire any of these locks explicitly with the `LOCK` command.

Remember that all of these lock modes are table-level locks, even if the name contains the word *row*; the names of the lock modes are historical. To some extent the names reflect the typical usage of each lock mode — but the semantics are all the same. The main difference between one lock mode and another is the set of lock modes with which each conflicts.

Two transactions cannot hold locks of conflicting modes on the same table at the same time. However, a transaction never conflicts with itself. For example, it may acquire `ACCESS EXCLUSIVE` lock and later acquire `ACCESS SHARE` lock on the same table.

Non-conflicting lock modes may be held concurrently by many transactions.

## Lock Modes (Cont)

Lock Mode	SQL Commands	Conflicts With
SHARE ROW EXCLUSIVE		ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE
EXCLUSIVE	DELETE, UPDATE	ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE
ACCESS EXCLUSIVE	ALTER TABLE, DROP TABLE, TRUNCATE, REINDEX, CLUSTER, VACUUM FULL	ACCESS SHARE, ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE, EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

90

## Lock Modes (Continued)

## Checking for Lock Conflicts

Lock conflicts can be:

- Verified by querying `pg_locks`
- Resolved by an administrator
- Caused by:
  - Concurrent transactions accessing the same object
  - Resource queue locks
  - Transaction deadlocks between segments (rare)

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

91

When an object is locked, subsequent concurrent queries that try to access the object need to wait for the lock to be freed. This can appear to the end user or application like the query is *hanging* or not being processed.

You should be aware of the following types of conflicts:

- **Lock Conflicts** – As long as a deadlock situation is not detected, a transaction seeking either a table-level or row-level lock will wait indefinitely for conflicting locks to be released. It is therefore a bad idea for applications to hold transactions open for long periods of time, such as while waiting for user input.
- **Queue Locks** – If a query is submitted through a resource queue that has exceeded its limits, the queue is locked until resources are freed. The `pg_locks` table will show a lock on the queue object for a particular query and session.
- **Deadlocks** – PostgreSQL has a built-in deadlock detector that catches most deadlock situations when the query is submitted. However, Greenplum does not have a global deadlock detector across all segments in the system. Deadlock detection is local at the segment level. There are rare cases where a transaction deadlock can occur because of a deadlock at the segment level.

## Checking for Lock Conflicts Example



### Example: Determine which query has a lock

```
SELECT locktype, database, c.relname, l.relation,
       l.transactionid, l.transaction, l.pid, l.mode, l.granted,
       a.current_query
  FROM pg_locks l, pg_class c, pg_stat_activity a
 WHERE l.relation=c.oid AND l.pid=a.procpid
 ORDER BY c.relname;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

92

Checking the pg\_locks view and joining it with pg\_class, which lists the objects involved, and pg\_stat\_activity, which lists the users and sessions involved, can let an administrator know what is going on and identify lock conflicts. If a conflict is detected, killing the process id of a conflicting query on the master resolves the problem.

## Lab: Data Manipulation Language and Data Query Language

In this lab, you use DML and DQL to access and manage data in the tables.

You will:

- Insert, update, and delete records
- Access data to generate a report

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

93

In this lab, you use DML and DQL to access and manage data in the tables.

## Module 4: Defining and Securing the User Database

### Lesson 2: Summary

During this lesson the following topics were covered:

- Support for SQL statements in the Greenplum Database
- Concurrency control
- Functions and operators to build queries
- Transaction concurrency in the Greenplum Database

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

94

This lesson covered the DML and DDL syntax used to access and manipulate database objects, including how to insert, delete, and update data in tables, using functions, and various other statements for accessing database objects. Concurrency control allows both the system and users to lock database object to ensure that data retrieved is valid, despite changes made to the objects. Users create functions and use operators to repetitively access commonly used syntax.

# Module 4: Defining and Securing the User Database

## Lesson 3: Roles, Privileges, and Resources

In this lesson, you examine the concepts of roles, privileges, and workload management in Greenplum and identify how to manage resource queues.

Upon completion of this lesson, you should be able to:

- Define roles and privileges
- Describe which roles get object privileges
- Identify security issues that can affect your database and corresponding data
- Create roles with specific privileges to control access
- Isolate users at the physical and functional layers
- Describe the workload management process

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

95

To provide your users access to the database and to ensure each user's data is protected from unauthorized view, you must define roles for the database. You must also manage resources to ensure that users and their queries do not hog system resources.

In this lesson, you will:

- Define roles and privileges for the Greenplum Database system.
- Identify the privileges that can be assigned to specific roles.
- Identify security issues that can affect your database and corresponding data.
- Examine privileges that can be used to enforce access.
- Identify practices designed to isolate users.
- Examine the workload management processes and how to manage resources.

## Roles and Privileges Overview

Roles:

- Can be a user, group, or both
- Are not related to OS users and groups
- Use attributes to determine permission levels
- Are given access privileges to database objects
- Can be members of other roles
- Are defined at the system-level

Every system has a default superuser role

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

96

Greenplum Database 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.

Greenplum Database roles:

- Are separate from the users and groups managed by the operating system on which the server runs. For your convenience, you may want to maintain a relationship between operating system user names and Greenplum Database role names, as many of the client applications use the current operating system user names.
- Have attributes that determine what their permission levels are.
- Can own database objects, such as tables, and can assign privileges on those objects to other roles to control access to the objects. The role that creates an object becomes its owner by default.
- Can be members of other roles, thus a member role can inherit the attributes and privileges of its parent role.
- Are defined at the system level, meaning they are valid for all databases in the system. Managing roles and privileges in Greenplum Database is the same as in PostgreSQL.

Every freshly initialized Greenplum Database system contains one predefined role. This role is a superuser role. By default it has the same name as the operating system user that initialized the Greenplum Database system. In our example, the role is `gpadmin`. Before creating more roles, you first have to connect to Greenplum the superuser role.

## Role Privileges

A user account:

- Has login privileges
- Is automatically assigned the following default attributes:
  - **NOSUPERUSER**
  - **NOCREATEDB**
  - **NOCREATEROLE**
  - **INHERIT**
  - **NOLOGIN** (must explicitly give `LOGIN` to user-level roles)

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

97

A user-level role is a role that can log in to the database and issue commands. By default, when you create a new role without declaring any attributes, it is assigned the following permissions:

- **NOSUPERUSER** – No superuser (a superuser is someone who have complete permissions and privileges to everything – should be given with caution)
- **NOCREATEDB** – Cannot create databases
- **NOCREATEROLE** – Cannot create other roles
- **INHERIT** – Does inherit the permissions and privileges of the roles it's a member of
- **NOLOGIN** – No login capabilities (must explicitly give this to user accounts)

While you can create roles with `LOGIN` privileges using the `CREATE USER` command, this command is deprecated in PostgreSQL 8.1. The recommended method is to use the `CREATE ROLE` command. You can also use the Greenplum client, `createuser`, to create a role.

You may create nested roles, where a role is a parent to another role. However, you should not have nested users, or users who are parents to other roles.

## Roles – Superusers

A superuser:

- Bypasses all permission checks
- Should not be used for daily administration

Create the following administrative roles to work with:

- SUPERUSER      • CREATEROLE
- CREATEDB      • PASSWORD



**Note:** It is good practice to create a role that has the CREATEDB and CREATEROLE privileges, but is not a superuser. Use this role for all routine management of databases and roles. This approach avoids the dangers of operating as a superuser for tasks that do not require it.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

98

A database superuser bypasses all permission checks.

This is a dangerous privilege and should not be used carelessly. It is best to do most of your work as a role that is not a superuser.

To accomplish this, complete the following tasks:

- Create a database superuser called SUPERUSER – Use the command `CREATE ROLE name SUPERUSER` to create a SUPERUSER role. You must do this as a role that is already a superuser.
- Create a role that can create databases – The new role must be explicitly given permission to create databases. To create such a role, use the command, `CREATE ROLE name CREATEDB`.
- Create a role that can create roles – A role must be explicitly given permission to create more roles. To create such a role, use `CREATE ROLE name CREATEROLE`.
- A role with CREATEROLE privilege can alter and drop other roles, too, as well as grant or revoke membership in them. However, to create, alter, drop, or change membership of a superuser role, superuser status is required; CREATEROLE is not sufficient for that.
- Assign a password – A password is only significant if the client authentication method requires the user to supply a password when connecting to the database. The password, md5, and crypt authentication methods make use of passwords. Database passwords are separate from operating system passwords. Specify a password upon role creation with `CREATE ROLE name PASSWORD 'string'`.

## Common Role Attributes

Role Attribute	Description
SUPERUSER   NOSUPERUSER	Determines if the role is a superuser. You must yourself be a superuser to create a new superuser. NOSUPERUSER is the default.
CREATEDB   NOCREATEDB	Determines if the role is allowed to create databases. NOCREATEDB is the default.
CREATEROLE   NOCREATEROLE	Determines if the role is allowed to create and manage other roles. NOCREATEROLE is the default.
INHERIT   NOINHERIT	Determines whether a role inherits the privileges of roles it is a member of. INHERIT is the default.
LOGIN   NOLOGIN	Determines whether a role is allowed to log in. NOLOGIN is the default.
CONNECTION LIMIT <i>connlimit</i>	If role can log in, this specifies how many concurrent connections the role can make. -1 (the default) means no limit.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

99

The table lists common user attributes you use to enforce security.

## Common Role Attributes (Cont)

Role Attribute	Description
PASSWORD <i>'password'</i>	Sets the role's password. A null password can optionally be written explicitly as PASSWORD NULL.
ENCRYPTED   UNENCRYPTED	Controls whether the password is stored encrypted in the system catalogs. The default behavior is determined by the configuration parameter password_encryption (currently set to MD5).
VALID UNTIL <i>'timestamp'</i>	Sets a date and time after which the role's password is no longer valid. If omitted the password will be valid for all time.
RESOURCE_QUEUE <i>queue_name</i>	Assigns the role to the named resource queue for workload management.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

100

## Common Role Attributes (Continued)

## SQL Commands for Roles

Use the following SQL commands and Greenplum applications to manage roles:

Action	SQL Syntax	Greenplum Application
Create a role	CREATE ROLE	createuser
Drop a role	DROP ROLE	dropuser
Alter a role	ALTER ROLE	

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

101

You can use the SQL commands, CREATE ROLE, DROP ROLE, and ALTER ROLE to manage users. The Greenplum applications, createuser and drop user can also be used to manage roles. Only superusers with CREATEROLE privileges can create roles, and so these SQL commands or Greenplum applications must be invoked by a user role with appropriate privileges, such as the superuser role.

## Roles and Privileges – Example

### Example: Create roles with the LOGIN privilege

```
CREATE ROLE john WITH LOGIN;  
CREATE USER john;
```

### Example: Change a role and assign it CREATEDB privileges

```
ALTER ROLE john WITH CREATEDB;
```

### Example: Grant read access to the gphdfs protocol

```
GRANT SELECT ON PROTOCOL gphdfs TO john
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

102

The first example on the slide shows two roles being created, both with login privileges.

The second example shows the ALTER ROLE command being used to change permission attributes for a user.

The last example has the SELECT access being granted to the role, john, so that the role can access it, such as when creating a read-only external table using the gphdfs protocol.

## Role Membership or Groups

A role:

- Can be a member of other roles
- Inherits object privileges of the parent role
- Allows you to set object privileges in one place
- Will not inherit:
  - LOGIN
  - SUPERUSER
  - CREATEDB
  - CREATEROLE
- Can use `SET ROLE` to obtain privileges

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

103

It is frequently convenient to group users together to ease management of permissions and privileges. This allows you to grant or revoke privileges for a group of users. In PostgreSQL this is done by creating a role that represents the group and granting membership in the group role to individual user roles.

The role attributes `LOGIN`, `SUPERUSER`, `CREATEDB`, and `CREATEROLE` are never inherited as ordinary privileges on database objects. Using the `SET ROLE` command, you must assign a role that already has these attributes defined.

In the example on the slide, the `admin` role is granted `CREATEDB` and `CREATEROLE` attributes. If the user `sally` is a member of the `admin` role, she can issue the following command to assume the role attributes of the parent role: `SET ROLE admin;`

## Role Membership or Groups – Examples

To manage access to roles:

- Use GRANT command to grant membership
- Use REVOKE command to remove a member from a role



### Example: Grant and revoke privileges

```
CREATE ROLE admin CREATEROLE CREATEDB;  
GRANT admin TO john, sally;  
REVOKE admin FROM bob;  
SET ROLE admin;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

104

Once the group role exists, you can add and remove members (user roles) using the GRANT and REVOKE commands. Those user roles then inherit the attributes and privileges of the parent role.

In the example shown, the `admin` role is assigned to two roles, `john` and `sally`. Both roles now inherit the `CREATEROLE` and `CREATEDB` privileges. In the last line of the example, the current session user has been changed to `admin`.

## Object Privileges

Objects:

- Are owned by object creator
- Can be made accessible to other roles
- Can be made accessible to all through the `PUBLIC` role
- Are managed with `GRANT` and `REVOKE` commands
- Assigned to deprecated roles are managed with `DROP OWNED` and `REASSIGN OWNED`

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

105

When an object, such as a table, is created, it is assigned to an owner. The owner is normally the role that created the object. This allows the role completely manage this object.

The owner role is given all privileges on that object and can assign privileges to other roles. The owner, or a superuser, are the only roles that can drop the object.

`PUBLIC` is a special predefined role that includes all roles, even ones that are created later.

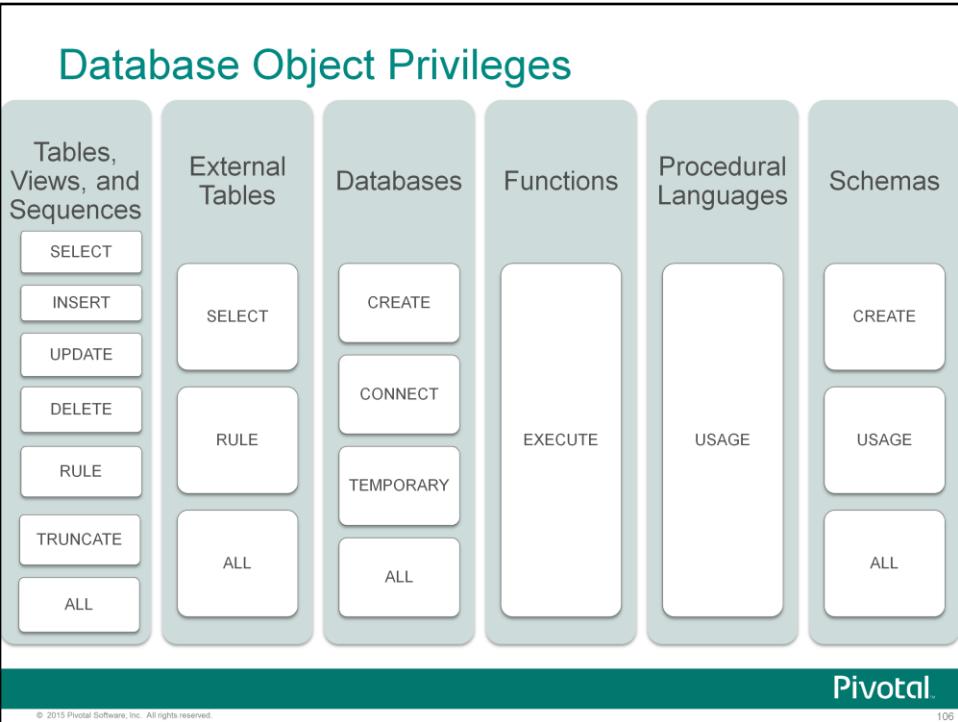
Use the following commands to manage objects owned by deprecated roles:

- `REASSIGN OWNED` reassigns all the objects in the current database that are owned by *old\_role* to *new\_role*. Note that it does not change the ownership of the database itself.
- `DROP OWNED` drops all the objects in the current database that are owned by one of the specified roles. Any privileges granted to the given roles on objects in the current database will also be revoked.

**Note:** Privileges are not passed down in inherited table structures from parent to child tables. Must be explicitly set on all tables.

You must grant access to an object if other users wish to use it. For managing object privileges, you grant the appropriate permissions to the group-level role only.

The member user roles then inherit the object privileges of the group role.



106

These are the database objects and associated privileges that are relevant in Greenplum Database. Privileges may vary based on the object type. Privileges provide access rights to the named objects, allowing roles to read, access, or modify the object. Privileges are not hierarchical. For example granting **ALL** on a database does not include **ALL** on the tables in that database. Those privileges must be assigned individually.

## Object Privileges – Examples

### Example: Grant permissions to admin

```
GRANT ALL ON DATABASE  
mydatabase TO admin  
WITH GRANT OPTION;
```

### Example: Assign all of one user's objects to another user

```
REASSIGN OWNED BY sally TO bob;
```

### Example: Grant SELECT to PUBLIC

```
GRANT SELECT ON TABLE  
mytable TO PUBLIC;
```

### Example: Drop all objects owned by visitor

```
DROP OWNED BY visitor;
```

### Example: Remove INSERT and UPDATE

```
REVOKE INSERT UPDATE ON TABLE  
mytable FROM sally;
```

Pivotal.

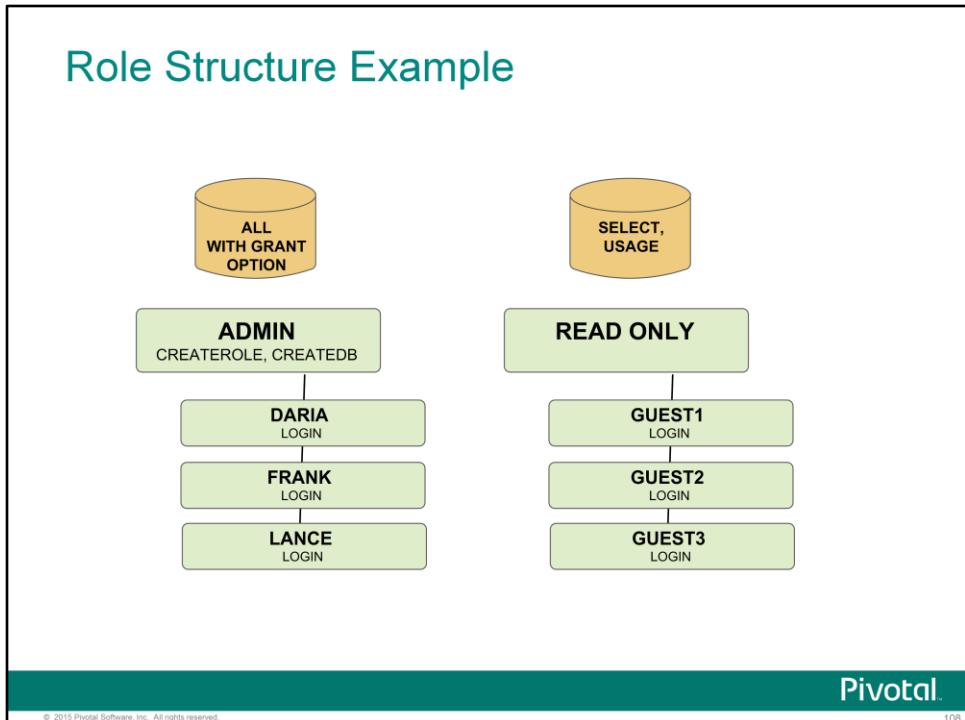
© 2015 Pivotal Software, Inc. All rights reserved.

107

Here are some examples of how to alter privileges for users and objects:

- In the first example, all privileges are granted to the user role, `admin`, for the database `mydatabase`. The `admin` role is also allowed to grant permissions to other roles for the specified database.
- The second example on the left side shows only `SELECT` privileges on the table, `mytable`, are granted to the role, `public`.
- The third example on the left side of the slide shows both `INSERT` and `UPDATE` permissions on the table, `mytable`, being revoked for the role, `sally`.
- The first example on the right side of the slide shows that all objects owned by one role, `sally`, are being reassigned to another role, `bob`.
- In the last example on the slide, all objects owned by the role, `visitor`, are removed from the database.

## Role Structure Example



On this slide is an example of two role hierarchies you can define using the following specifications:

- The first is a top level role named `ADMIN` with permissions to create roles and databases.
- The second is a role called `read-only` with no create permissions.
- Each Group level role has login accounts associated with it. Note that the permission attributes are assigned at the parent level.
- Privileges are assigned at the parent-level role. For the admin role, you can assign all privileges to all database objects and also give the grant option so members of that role could grant privileges to other roles.
- For the read-only role, assign the `SELECT` privileges on tables. You can also assign `USAGE` on schemas.

## System-level Security

Problems encountered with system security:

- Pessimistic model is used, so users do not have access to anything unless specifically assigned.
- Sharing accounts makes auditing difficult
- Users are tempted to run additional workloads on master node, such as ETL
- Once logged into the master, all other hosts are available since SSH keys are exchanged
- Network infrastructure is not under our control
- System administration is not under our control

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

109

In system level security, the concern is about defining privileges for users who access the environment. Unauthorized users with access to the system increase their chances of gaining access to data.

Problems that are oftentimes encountered with securing a system include:

- Pessimistic model prevents users from having access to specific items unless specifically assigned. While there is a benefit in that, as roles are implemented at the system level, it can that a user is prevented from performing tasks to which they have been assigned.
- Account sharing can make auditing a difficult task. It is not easy to determine who did what, when, and who approved the action if one or two major accounts are used by multiple individuals.
- Users may be tempted to hog resources and run additional workloads on the master, including data loads through ETL. This can freeze other users out of resources until that action is completed.
- Once logged into the master, all other hosts are available to the user on the master node since SSH keys have been exchanged.
- Network security issues outside of the control of the database can have an adverse impact.
- Other system administration outside of the database can also have an adverse impact.

## Database Security

Problems relating to database security include:

- `gpadmin` user has access to everything
- Pessimistic model is used; users do not have access to anything unless specifically assigned privileges
- Unless resource queues are configured, there are no limits on what users can run
- Default `pg_hba.conf` uses trust instead of enforcing passwords
- Customers are tempted to share user accounts

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

110

Problems that are specifically associated with the database include the following:

- The superuser role created on database initialization has access to every aspect of the system. This can be dangerous particularly if the role is used for daily administration and is shared by multiple users.
- Pessimistic model usage in the database has benefits and detractors. By employing a pessimistic model, users do not have access until specifically assigned. However, if the correct privileges are not assigned, this can cause frustration and delay for that user.
- There are no limits on resources unless specifically configured. Once again, a user can hog resources and effectively, shut other users out of the database.
- The default configurations in the `pg_hba.conf` are loosely configured to use trust. This bypasses security measures that should be in place to require authentication from the connecting user.
- As with regular system-level security, users may be tempted to share user accounts. Users, or roles, can be defined for each user to protect their data. If multiple users share the same role, this reduces the chances of finding who had access to the system and to data.

## Database Security – Roles

Roles can be used to enhance security with the following:

- Each person should have their own role
- Roles should be further divided into groups, which are usually roles that do not have the `LOGIN` attribute
- Privileges should be granted at the group level whenever possible
- Privileges should be as restrictive as possible
- Column level access can be accomplished with views
- Roles are not related to OS users or groups

Pivotal

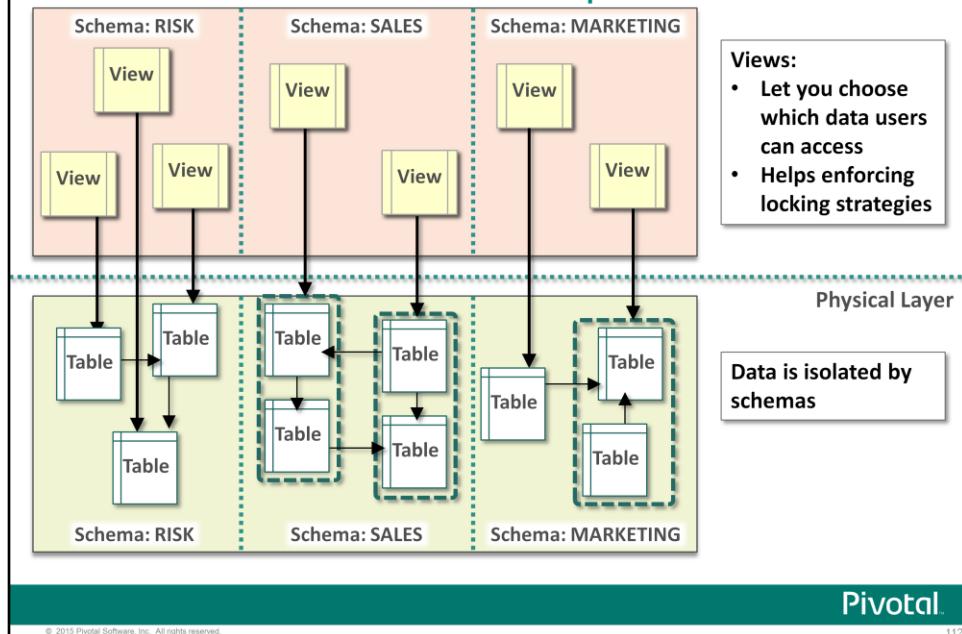
© 2015 Pivotal Software, Inc. All rights reserved.

111

You can implement the following rules to tighten security using roles:

- Roles are implemented at the system-level and are separate from operating system users. Each person that connects to the database should have their own role assigned to them. Users are roles with login attributes associated with the role. By assigning a role to each individual user, you increase the chances of properly auditing the system to ensure that the appropriate users have access to appropriate data. Application services should also be given a distinct role in the environment.
- Roles should be divided into groups to ease administration tasks. Groups represent role membership, where role is created and membership to that role is granted. The role can be assigned privileges which are inherited by members of the role. This makes it easier to assign privileges to users.
- By assigning privileges at the group level, you reduce the chances that an individual user may have a privilege that should have been revoked. You can also more quickly assign a group of privileges to users by adding them to the appropriate group.
- While the pessimistic model is in use, assigned privileges should be as restrictive as possible. Do not assign superuser privileges to a user who only needs to create roles.
- You can restrict access to data by implementing views instead of providing full access to tables.
- You can make it harder for someone to *guess* what a valid account is by removing the relationship between operating system users and groups to Greenplum roles. If you use the same names, you increase the chances that someone can guess role names.

## Database Architecture – Separation and



Users are isolated at the physical layer through the use of views. By isolating users, you:

- Insulate users from changes made to tables
- Control access to columns and rows
- Enforce locking strategies

Data is isolated by separating functional areas into schemas. By doing this, you:

- Simplify security
- Simplify backups and restores to data

## Security Example

The following is an example of user and role creation and assignment and how to use the roles to limit or grant privileges to user roles:

### Example: Inherit privileges through nested roles

```
CREATE ROLE batch;
GRANT select, insert, update, delete
    ON dimensions.customer TO batch;
CREATE ROLE batchuser LOGIN;
GRANT batch TO batchuser;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

113

In this example:

- The newly created role, `batch` is granted `SELECT`, `INSERT`, `UPDATE`, and `DELETE` privileges on the table `dimensions.customer`.
- The role is not a user as it was not assigned the `LOGIN` attribute.
- The `batchuser` role is then created with the `LOGIN` attribute and granted access to the `batch` role.
- The user, `batchuser`, can now has the privileges assigned to the role, `batch`.
- The object that the role is given access to is fully qualified, indicating the role has access only to the `customer` table in the `dimensions` schema.

## Greenplum Workload Management

Let us examine the following:

- What is workload management?
- How do you create resource queues?
- How do you assign roles to resource queues?
- What is the runtime evaluation of resource queues?
- What are resource queue configuration parameters?
- How do you view the status of resource queues?

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

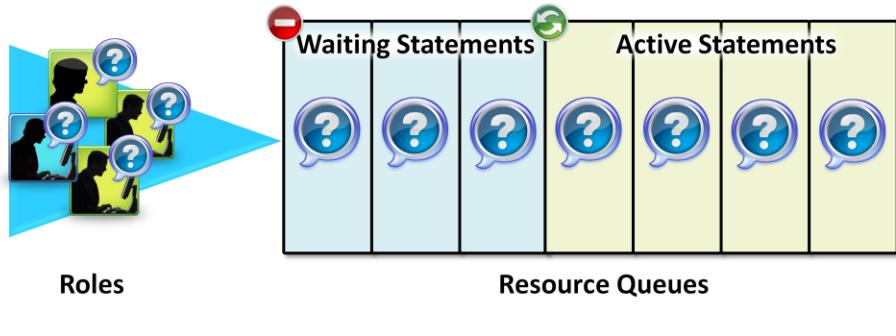
114

In this next section, you examine workload and resource management, a feature that gives you the ability to more closely limit the number of active queries in the system at any given time.

## Workload Management Overview

Workload management:

- Is used to limit the number of active queries
- Is meant to prevent overloading of system resources
- Looks at resource allocation from statement-level point-of-view



© 2015 Pivotal Software, Inc. All rights reserved.

115

The purpose of Greenplum Database workload management is to limit the number of active queries in the system at any given time in order to avoid exhausting system resources such as memory, CPU, and disk I/O.

This is accomplished by creating role-based *resource queues*. A resource queue has attributes that limit the size and/or total number of queries that can be executed by the users (or roles) in that queue.

By assigning all of your database roles to the appropriate resource queue, administrators can control concurrent user queries and prevent the system from being overloaded.

Administrators create resource queues for the various types of workloads in their organization. For example, you may have a resource queue for power users, web users, and management reports. The administrator would then set limits on the resource queue based on his estimate of how resource-intensive the queries associated with that workload are likely to be. Database roles, which include users and groups, are then assigned to the appropriate resource queue. A resource queue can have multiple roles, but a role can only be assigned to a single resource queue.

Starting with Greenplum 4.1, the Greenplum memory management system looks at allocating memory and CPU resources not from an operator or user point of view, but instead from the statement level point of view.

## Configurable Limits on a Queue

**Active Statement Count**

This parameter determines query entry into the system to actively run.

The maximum number of queries executing in the active queue is limited by the `active_statements` limit.

**Active Statement Memory**

This parameter determines resources allocated to the query.

Each query submitted to the queue will be given a portion of the memory. This amount is based on the amount of queries that can run in the queue or the cost of the query.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

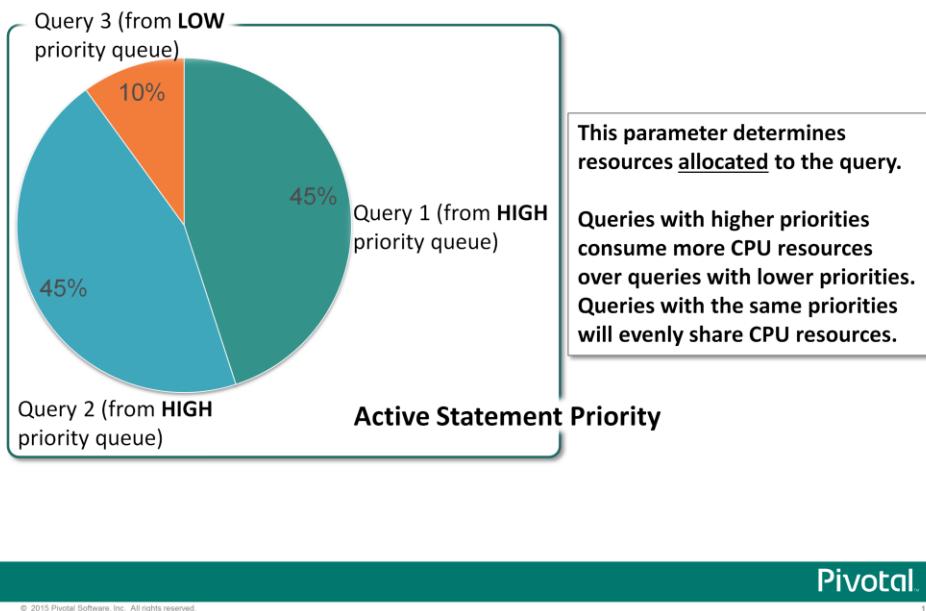
116

Resource queues can be created to address different types of workloads, such as workloads for power users, web users, and management reports. A queue for a web user may be fast and require very little resources, whereas the workload for management reports may be more intensive, requiring more resources, such as memory.

You can set limits on a queue using the following configurable limits:

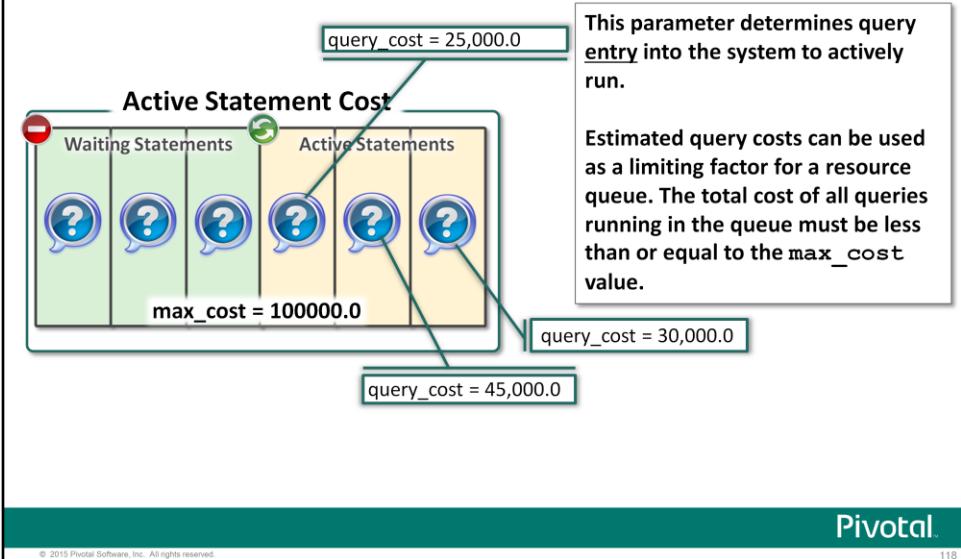
- **Active statement count** – This limit allows you to set the maximum number of statements that can run concurrently. The total number of queries running in a resource queue with an active statement limitation must be less than or equal to the defined ceiling. If the maximum number of queries are running, an additional query that enters the queue must wait for one of the active queries to complete before being allowed to execute on the system.
- **Active statement memory** – The active statement memory limit lets you set the total amount of memory that all queries assigned to the resource queue can consume. No additional memory will be allocated to the queue once the maximum has been reached and each query is allocated the same amount of memory, which is the total available memory divided by the number of active queries. It is recommended you use this limit in conjunction with the active statement count limit.

## Configurable Limits on a Queue (Cont)



- **Active statement priority** – The active statement priority queue lets you define the relative priority of the queue to other queues in terms of available CPU resources. All queries entering the queue are given the same priority. If the priority of one resource queue is higher than the priority on another resource queue, the higher priority queue will consume more CPU resources.

## Configurable Limits on a Queue (Cont)



- **Active statement cost** – The query planner estimates a cost for each query submitted. You can set the maximum query planner cost for a resource queue, limiting the number of queries that can enter the queue by the total cost of all queries in the queue. The cost is measured in units of disk page fetches, where 1.0 is equal to one sequential disk page read.

## Creating Resource Queues

To create resource queues and manage thresholds, use the following commands:

Action	SQL Syntax
Create or alter a resource queue	<code>CREATE ALTER RESOURCE QUEUE <i>name</i> WITH (MAX_COST=<i>n</i>   ACTIVE_STATEMENTS=<i>n</i> [, COST_OVERCOMMIT] [,MIN_COST=<i>n</i>] [,MEMORY_LIMIT='<i>nM GB</i>' ] [,PRIORITY=<i>level</i>])</code>
Drop a resource queue	<code>DROP RESOURCE QUEUE <i>name</i></code>

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

119

Greenplum has added the SQL commands `CREATE RESOURCE QUEUE`, `ALTER RESOURCE QUEUE`, `DROP RESOURCE QUEUE` which you will use to create and manage the limits of your resource queues. Queries gain entry into the system so that they can actively run with the following resource queue limits:

- **ACTIVE\_STATEMENTS** – Limits the number of queries that can be executed by roles assigned to that queue.
- **MAX\_COST** – Limit the total size of queries that can be executed by roles assigned to that queue based on query cost. Cost is measured in the estimated total cost for the query as determined by the Greenplum query planner. Therefore, an administrator must be familiar with the queries typically executed on the system in order to set an appropriate cost threshold for a queue. Cost is measured in units of disk page fetches; 1.0 equals one sequential disk page read.

The following additional settings can be used in conjunction with `ACTIVE_STATEMENTS` and `MAX_COST` to determine how much resources queries receive once they are actively running:

- **MEMORY\_LIMIT** – Setting this limit sets the total memory quota for all queries within the resource queue. Each query is allocated a pre-determined amount of memory.

## **Creating Resource Queues (Continued)**

- **COST\_OVERCOMMIT** – Queries that exceed the cost threshold can continue executing, provided there are no other queries in the system at the time the query is submitted.
- **MIN\_COST** – The query cost limit of what is considered a small query. Queries with a cost under this limit will not be queued but will instead run immediately.
- **PRIORITY** – The priority does not determine which queries enter the active queue, but instead decides how much CPU resources the query receives. Queries with higher priority receive a larger share of available CPU resources, decreasing the CPU resources available to any in-flight queries with lower priorities.

## Managing CPU Utilization with Priorities

Priorities:

- Control a resource queue's consumption of resources
- Are managed with the following SQL commands:

Action	SQL Syntax
Create a resource with a specific priority	<code>CREATE RESOURCE QUEUE name WITH (PRIORITY=level)</code>
Alter a resource queue to have a specific priority	<code>ALTER RESOURCE QUEUE name WITH (PRIORITY=level)</code>
<ul style="list-style-type: none"><li>• Can be one of the following:<ul style="list-style-type: none"><li>— MIN</li><li>— HIGH</li><li>— LOW</li><li>— MAX</li><li>— MEDIUM (default setting if not specified)</li></ul></li></ul>	

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

121

To control a resource queue's consumption of available CPU resources, you can assign an appropriate priority level. When high concurrency causes contention for CPU resources, queries and statements associated with a high-priority resource queue will claim a larger share of available CPU than lower priority queries and statements.

Priority settings are created or altered using the `WITH` parameter of the commands `CREATE RESOURCE QUEUE` and `ALTER RESOURCE QUEUE`.

You can assign priorities with the following levels:

- **MIN** – This is the lowest possible priority setting for a resource queue.
- **LOW** – At one time, the lowest possible available setting, resource queues with this level consume great CPU resources than those with a level of **MIN**.
- **MEDIUM** – This is the default level and is automatically assigned when a `PRIORITY` is not defined for the resource queue.
- **HIGH** – Queries with this setting take precedence over those assigned to **MEDIUM** or **LOW** resource queues.
- **MAX** – This is the highest level that can be assigned to a resource queue. Queries with this setting take precedence over all others.

## Managing Memory Allocation with Memory Limits

Query memory allocation is controlled with:

- `MEMORY_LIMIT` parameter for the resource queue:

Action	SQL Syntax	Memory Allocation per Query
Create or alter a resource with a specific memory limit and maximum number of statements	<code>CREATE ALTER RESOURCE QUEUE <i>name</i> WITH (ACTIVE_STATEMENTS = <i>x</i>, MEMORY_LIMIT='<i>xxxM GB</i>' )</code>	<code>MEMORY_LIMIT / ACTIVE_STATEMENTS</code>
Create or alter a resource queue to have a specific memory limit with <code>MAX_COST</code> defined	<code>CREATE ALTER RESOURCE QUEUE <i>name</i> WITH (MAX_COST = <i>x.x</i>, MEMORY_LIMIT='<i>xxxM GB</i>' )</code>	<code>MEMORY_LIMIT * (query_cost / MAX_COST)</code>

- At the database level with `statement_mem` and `max_statement_mem`

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

122

Memory can be controlled both within a resource queue as well as at the database level.

For a resource queue, a maximum amount of memory is allocated to the queue with the `MEMORY_LIMIT` parameter and value as part of the `CREATE RESOURCE QUEUE` command. The value specified can be in Megabytes or Gigabytes.

Each query is allocated a certain amount of memory depending on how the queue is configured:

- If the resource queue is created with memory limits and `ACTIVE_STATEMENTS` set to a specified value, each query will be allocated an equal amount of memory. The memory allocated is determined by the `MEMORY_LIMIT` value divided by the `ACTIVE_STATEMENTS` value.
- If the resource queue is created with memory limits and `MAX_COST` set to a specified floating value, the query is allocated memory based on the formula, `MEMORY_LIMIT * (query_cost / MAX_COST)`.

The resource queue allocation can be overridden by the use of the `statement_mem` server configuration parameter. A query submitted to the environment after `statement_mem` is set will be allocated the amount of memory defined in `statement_mem`, as long as the amount allocated is less than the value defined in `max_statement_mem`.

One thing to note, the memory limit defined across all resource queues must be less than the total physical memory available to a segment host.

## Creating Resource Queues – Examples

### Example: Create a resource queue with up to 3 active queries

```
CREATE RESOURCE QUEUE adhoc WITH  
(ACTIVE_STATEMENTS=3);
```

### Example: Create a resource queue with a planner cost limit

```
CREATE RESOURCE QUEUE webuser WITH  
(MAX_COST=100000.0);
```

### Example: Create a resource queue with a minimum cost limit

```
CREATE RESOURCE QUEUE adhoc WITH  
(ACTIVE_STATEMENTS=10, MIN_COST=100.0);
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

123

In the first example, for all roles assigned to the `adhoc` resource queue, only three active queries can be running on the system at any given time. If this queue has three queries running, and a fourth query is submitted by a role in that queue, that query must wait until a slot is free before it can run.

In the second example, a resource queue named `webuser` is created with a query cost limit of `100000.0` (`1e+5`). For all roles assigned to the `webuser` resource queue, it will only allow queries into the system until the planner cost limit of `100000.0` is reached. If this queue has 200 queries with a `500.0` cost all running at the same time, and query 201 with a `1000.0` cost is submitted by a role in that queue, that query must wait until space is free before it can run.

In the third example, any queries that fall below the defined `MIN_COST` limit will run immediately. The restriction of ten active queries still applies however.

## Assigning Roles to Resource Queries

Resource queues:

- Must be assigned at the user-level (group-level ignored)
- Do not affect superuser roles
- Are managed with the following SQL commands:

Action	SQL Syntax
Add a resource queue to a role	<code>ALTER ROLE role_name RESOURCE QUEUE queue_name;</code>
Create a role and assign it to a resource queue	<code>CREATE ROLE role_name WITH LOGIN RESOURCE QUEUE queue_name;</code>



**Note:** All users are assigned to a resource queue, even if one is not specified. The default resource queue is pg\_default.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

124

Currently resource queues must be assigned on a user-by-user basis. If you have a role hierarchy such as a group-level role, then assigning a resource queue to the group does not propagate down to the users.

All superuser roles are exempt from resource queue limits. Even if you assign a superuser role to a resource queue, their queries will always run regardless of restrictions defined for the resource queue.

To assign a user to a resource queue use the `CREATE ROLE` and `ALTER ROLE` commands. A user role can only be assigned to one resource queue.

## Runtime Evaluation of Resource Queues

For runtime evaluation:

- Resource queues are evaluated independently of each other
- Superusers (and unassigned roles) are exempt
- Queries are evaluated on first-in, first-out basis

If a query causes the queue to exceed its limits:

- Query must wait until queue resources are free
- Query must wait until queue is idle (`COST_OVERCOMMIT=TRUE`)
- Query will never run (`COST_OVERCOMMIT=FALSE`)

Evaluated SQL statements include:

- `SELECT, SELECT INTO, CREATE TABLE AS SELECT, DECLARE CURSOR`
- (optional) `INSERT, UPDATE, DELETE`

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

125

At runtime, when the user submits a query for execution, that query is evaluated against the resource queue's limits. If the query does not cause the queue to exceed its resource limits, then that query will run immediately. If the query causes the queue to exceed its limits, such as if the maximum number of active statement slots are currently in use, then the query must wait until queue resources are free before it can execute. If the queue is idle however, and `COST_OVERCOMMIT` is set to true, a query that exceeds the `MAX_COST` will be allowed to run. If `COST_OVERCOMMIT` is set to false, the query will never execute.

Queries submitted through a queue are evaluated and executed on a first in, first out basis.

Not all SQL statements submitted through a resource queue are evaluated against the queue limits. By default only `SELECT, SELECT INTO, CREATE TABLE AS SELECT, and DECLARE CURSOR` statements are evaluated. If the server configuration parameter `resource_select_only` is set to off, then `INSERT, UPDATE, and DELETE` statements will be evaluated as well.

## Resource Queue Configuration Parameters

The following parameters are used to configure resource queues:

Configuration Parameter	Default Value	Description
max_resource_queues	9	The maximum number of resource queues in the system
max_resource_portals_per_transaction	64	The maximum number of open cursors per transaction
resource_select_only	on	Determines which queries are managed by resource queues
stats_queue_level	off	Enables the collection of statistics on resource queue usage
resource_cleanup_gangs_on_wait	on	Clean up idle segment worker processes

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

126

Here are the parameters for resource queues shown with their defaults. All of these must be set in the master postgresql.conf and require server restart to take effect:

- **max\_resource\_queues** — This parameter sets the maximum number of resource queues that can be created in a Greenplum Database system. Note that resource queues are system-wide (as are roles) so they apply to all databases in the system.
- **max\_resource\_portals\_per\_transaction** — This parameter sets the maximum number of simultaneously open cursors allowed per transaction. Note that an open cursor will hold an active query slot in a resource queue.
- **resource\_select\_only** — This parameter sets the types of queries managed by resource queues. If set to on, then SELECT, SELECT INTO, CREATE TABLE AS SELECT, and DECLARE CURSOR commands are evaluated. If set to off INSERT, UPDATE, and DELETE commands will be evaluated as well.
- **stats\_queue\_level** — Setting this parameter to on enables the collection of statistics on resource queue usage, which can then be viewed by querying the pg\_stats\_resqueue system view. This is set to off by default.
- **resource\_cleanup\_gangs\_on\_wait** — Setting this parameter cleans up idle segment worker processes before taking a slot in the resource queue.

## Memory Utilization System Parameters

Configuration Parameter	Default Value	Description
gp_resqueue_memory_policy	eager_free	The query plan is divided into stages and Greenplum Database eagerly frees memory allocated to a previous stage at the end of that stage's execution, then allocates the eagerly freed memory to the new stage
statement_mem max_statement_mem	125 (MB) 2000 (MB)	statement_mem: Allocates segment host memory per query max_statement_mem: Sets the maximum memory limit for a query. (seghost_physical_memory) / (average_number_concurrent_queries)
gp_vmem_protect_limit	8192 (MB)	Upper memory boundary that all query processes can consume on a segment host
gp_vmem_idle_resource_timeout	18000 (Milliseconds)	If database session is idle for longer than the time specified, the session will free system resources (such as shared memory), but remain connected to the database
gp_vmem_protect_segworker_cache_limit	500 (MB)	If a query executor process consumes more than this configured amount, then the process will not be cached for use in subsequent queries after the process completes

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

127

The following parameters determine how memory is utilized at the database level:

- **gp\_resqueue\_memory\_policy** – The parameter enables Greenplum memory management features, where memory management is done at the statement level instead of at the operator level. The parameter can be set to `auto`, `off`, or `eager_free`. If set to `auto` or `eager_free`, the new memory management system will be used. Otherwise, if set to `none`, the system prior to Greenplum Database 4.1 will be used.
- **statement\_mem and max\_statement\_mem** – These parameters are used to both allocate memory to a query and set the maximum amount of memory that can be allocated. Once the `statement_mem` parameter has been set, any queries executed after will ignore the resource queue settings for memory allocation and use this set value.
- **gp\_vmem\_protect\_limit** – The parameter sets the upper boundary that all query processes can consume on a segment host. Queries that cause this limit to be exceeded will be cancelled. This parameter must be set at the local level for each segment instance.
- **gp\_vmem\_idle\_resource\_timeout and gp\_vmem\_protect\_segworker\_cache\_limit** – These two parameters are used to free memory on segments held by idle database processes. The first parameter sets the resource timeout value in milliseconds while the second sets the cache size in megabytes. If the second parameter is exceeded, the process that exceeded the cache will not be cached for subsequent queries.

## Viewing Status of Resource Queues

The following query system tables provide insight into resource queues:

System Table	Description
pg_resqueue	Resource queues and attributes
gp_toolkit.gp_resq_role	Role to resource queue assignments
pg_roles	
pg_locks	Queues that have waiting statements
pg_stat_activity	Process information about active and waiting queries

To view queue status and statistics, access the following tables:

System Table	Description
gp_resqueue_status	Queue limits, number of active and waiting queries
pg_resqueue_status	
pg_stat_resqueues	Queue statistics and performance over time

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

128

Currently there is no management interface for workload management and resource queues. To see status information about resource queues, you'll have to query and join these system tables. Your training guides have sample queries for finding out certain information about resource queues. You may want to save these queries as VIEWS in your production Greenplum Database system. Several system tables from the gp\_toolkit or pg\_catalog schemas are available to track resource queue behavior and activity.

The pg\_resqueue\_status system view allows administrators to see status and activity for a workload management resource queue. It shows how many queries are waiting to run and how many queries are currently active in the system from a particular resource queue.

If you want to track statistics and performance of resource queues over time, you can enable statistics collecting for resource queues. This is done by setting the following server configuration parameter in your master postgresql.conf file:

```
stats_queue_level = on
```

Once this is enabled, you can use the pg\_stats\_resqueue system view to see the statistics collected on resource queue usage. Note that enabling this feature does incur slight performance overhead, as each query submitted through a resource queue must be tracked. It may be useful to enable statistics collecting on resource queues for initial diagnostics and administrative planning, and then disable the feature for continued use.

## Lab: Roles, Privileges, and Resources

In this lab, you create and manage roles for the Greenplum Database. You create resource queues to manage the workload in the Greenplum Database system.

You will:

- Create roles that are users and roles that are groups
- Grant privileges on database objects to a group-level role
- Create a resource queue and assign a user role to this resource queue

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

129

In this lab, you create and manage roles for the Greenplum Database. You also create resource queues to manage the workload in the Greenplum Database system. You also implement security measures to tighten access privileges in the Greenplum Database.

## Lab: Controlling Access

In this lab, you create users and groups to control the level of access that users receive.

You will:

- Implement basic security at the group level

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

130

In this lab, you create and manage roles for the Greenplum Database. You also create resource queues to manage the workload in the Greenplum Database system. You also implement security measures to tighten access privileges in the Greenplum Database.

## Module 4: Defining and Securing the User Database

### Lesson 3: Summary

During this lesson the following topics were covered:

- Roles and privileges
- Role assignment to object privileges
- Security issues that can affect your database and corresponding data
- Creating roles with specific privileges to control access
- Isolating users at the physical and functional layers
- Workload management process

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

131

This lesson covered the roles and privileges assigned to roles to protect sensitive data from unauthorized roles and groups. Administrators can assign specific privileges to individual roles or group roles, where the privileges are inherited by users within those group roles. Additionally, workload management lets you assign resource profiles to roles. This allows users to share resources, reducing the chances that queries will be starved out of resources.

## Module 4: Summary

Key points covered in this module:

- Identified and managed database objects available in the Greenplum Database
- Used data manipulation language and data query language to access, manage, and query data
- Managed workload management processes by defining and managing roles, privileges, and resource queues
- Implemented system-level and database-level security

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

132

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

- Identify and manage database objects available in the Greenplum Database.
- Use data manipulation language and data query language to access, manage, and query data.
- Manage workload management processes by defining and managing roles, privileges, and resource queues.
- Implement system-level and database-level security in the Greenplum Database.

# Module 5: Data Loading and Distribution

This module describes how to load data into the Greenplum Database using several methods and how to use table partitioning to divide data into smaller portions.

Upon completion of this module, you should be able to:

- Differentiate among the various types of supported tables in Greenplum Database and build and maintain these objects
- Load data into a Greenplum database instance using external tables, parallel loading utilities, user defined protocol, and SQL COPY
- Use table partitioning to logically divide data into smaller parts

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

1

Greenplum Database provides support for several different types of tables, all with specific purposes to support database administrators meet the needs of their consumers. You should be able to differentiate among these tables and determine when to use each of them.

When getting ready to perform data loads, you must evaluate if and how the data should be partitioned and what commands, applications, or tools to use to load the data. Part of the design strategy for organizing your data will be to optimize for query. Table partitioning can help by allowing you to divide very large tables into more manageable portions. You must also understand the impact that the different methods of data loading can have on performance and ease of administration.

In this module, you will:

- Differentiate among the various types of supported tables in Greenplum Database and build and maintain these objects
- Load data into a Greenplum database instance using external tables, parallel loading utilities, and SQL COPY.
- Use table partitioning to divide tables into smaller parts.

## Module 5: Data Loading and Distribution

### Lesson 1: Implementing Table Storage Models, Compression, and Tablespaces

In this lesson, you examine various methods of loading data into Greenplum and examine some performance impacts.

Upon completion of this lesson, you should be able to:

- Describe how to create and apply tablespaces to database objects
- Differentiate among various types of supported tables
- Identify the supported table storage models
- List the various compression methods that can be applied to various tables

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

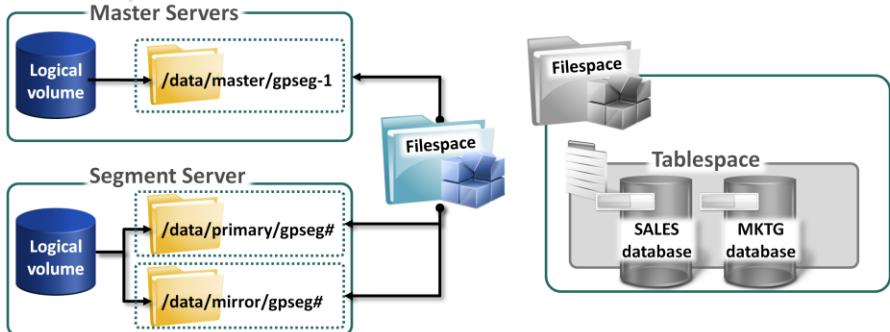
2

Greenplum offers several ways in which you can further manage user data within their own tablespaces and build appropriate tables to accommodate different types of data needs.

By the end of this lesson, you should be able to:

- Describe how to create and apply tablespaces to your database objects
- Differentiate among the various types of tables supported within Greenplum Database
- Identify the supported table storage models available to Greenplum Database tables
- List the various compression methods that can be applied to tables

## Filespace Review



- By default, the system filesystem, `pg_system`, is created on initialization
- All system relations are stored in the system filesystem by default
- All user relations are also stored in the system filesystem by default

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

3

A filesystem is a pointer to specific storage defined on your filesystem. The filesystem maps to a set of locations used by the master, standby, and segment hosts.

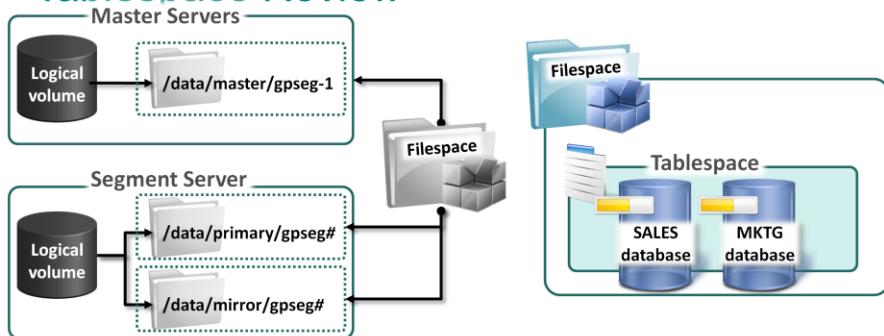
The system filesystem, `pg_system`, is automatically created on initialization and used to store system and user relations. In the example shown, the filesystem that supports the filesystem is associated with the `/data` directory. Data is stored as follows:

- Master and standby master store files in the `/data/master` directory.
- Primary segments store their files in `/data/primary` while mirror segments store their files in `/data/mirror`.

All tables and other database objects you subsequently create are stored within the system filesystem, unless otherwise specified.

Note that the filesystem is tied to a location on the filesystem and not necessarily to the filesystem root. You can have multiple filesystems share the same filesystem by pointing to different directories that exist on the filesystem.

## Tablespace Review



- Tablespaces *sit atop* filesystems interacting with the underlying filesystem
- A filesystem can support multiple tablespaces
- Two tablespaces are created on initialization: `pg_default` and `pg_global`

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

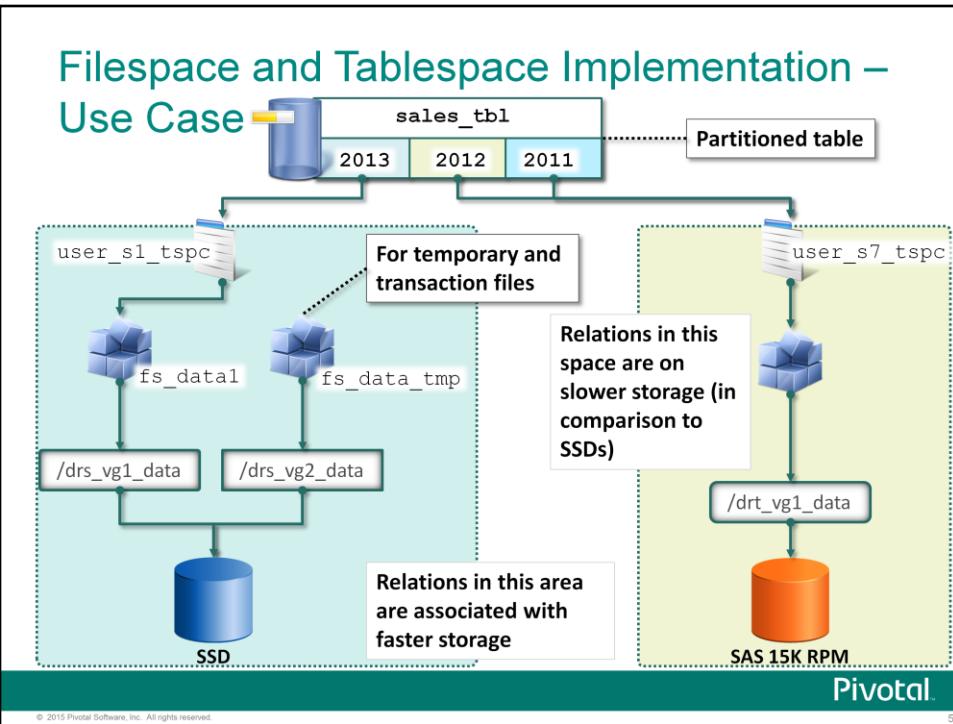
4

A tablespace interacts with the underlying filesystem by being associated with a specific filesystem. A filesystem can host multiple tablespaces, but a tablespace can only belong to a system filesystem. Tablespaces allow database administrators to effectively segregate data and take advantage of different storage profiles.

Within Greenplum, you can create and assign database objects, such as databases, tables, and indexes, to a specific tablespace.

Two tablespaces are defined by default:

- `pg_default` – This tablespace is the default tablespace used to store the default databases created for the system, `template0` and `template1`, as well as any other user-related database objects.
- `pg_global` – This tablespace is used to store shared system catalog for the environment.



Greenplum is not aware of, nor does it control the underlying filesystem. Instead, it relies on you creating the filesystem and associating the filesystem with the filesystem.

You can derive large benefits by maintaining multiple filesystems to meet the needs of customers within the environment. Your requirement may be to provide faster performance on data that is more often used, while relegating older, less-often used data to slower disks. By defining a filesystem on faster SSD disks for example, while defining another filesystem on SAS or SATA drives with lower speeds, you can help to meet the requirements of the service level agreement associated with the requirement.

In this example, partitions of a partitioned table, `sales_tbl`, are placed on separate tablespaces. The table is partitioned on the year field, with the latest year associated with the tablespace, `user_s1_tspc`, which is on the `fs_data1` filesystem. This filesystem is tied to a specific filesystem on faster SSD drives. The remaining two partitions are on one tablespace, which is associated with a slower disk device. As the partitions for previous years are less often used, they can be moved to tablespaces on slower devices, leaving the partition with newer data on the faster device.

A third filesystem in this slide, `fs_data_tmp`, is used to store temporary and transaction files. These files are created when queries are executed, during backup, and when data is stored sequentially. Moving these temporary and transaction files to a filesystem on faster storage can help to improve the performance of the database overall. This filesystem can be used to store user data. However, you cannot store temporary or transaction data across multiple filesystems – you are restricted to only one.

## Identifying Existing Filespaces and Tablespaces

List of all filesystems and their related filesystem locations

oid	fsname	fowner
3052	pg_system	10

(1 row)

List of the tablespaces, their associated filesystems, and the filesystem location

tblspc	filespc	seg_dbid	datadir
pg_default	pg_system	1	/data/master/gpseg-1
pg_default	pg_system	2	/data/primary/gpseg0
pg_default	pg_system	3	/data/primary/gpseg1
pg_default	pg_system	4	/data/mirror/gpseg0
pg_default	pg_system	5	/data/mirror/gpseg1
pg_default	pg_system	6	/data/master/gpseg-1
pg_global	pg_system	1	/data/master/gpseg-1
pg_global	pg_system	2	/data/primary/gpseg0
pg_global	pg_system	3	/data/primary/gpseg1
pg_global	pg_system	4	/data/mirror/gpseg0
pg_global	pg_system	5	/data/mirror/gpseg1
pg_global	pg_system	6	/data/master/gpseg-1

(12 rows)

Pivotal

You can determine the filesystems that exist within Greenplum by querying two system tables:

- `pg_filespace` lists the filesystems that exist within the environment.
- `pg_filespace_entry` lists the directories and the filesystem database IDs associated with each filesystem in the environment.

The OID listed for the `pg_filespace` table can be used to join the `pg_filespace` table to the `pg_filespace_entry` table on the `fsefsoid` column to list all directories associated with a specific filesystem.

The second sample shows all of the tablespaces defined in the system, along with their associated filesystem and filesystem. The tablespaces are listed within the `pg_tablespace` table. The syntax in the second sample is as follows:

```
SELECT      spcname as tblspc, fsname as filespc,
            fsedb as seg_dbid, fselocation as datadir
FROM        pg_tablespace pgts, pg_filespace pgfs,
            pg_filespace_entry pgfse
WHERE       pgts.spcfsoid=pgfse.fsefsoid
            AND pgfse.fsefsoid=pgfs.oid
ORDER BY    tblspc, seg_dbid;
```

# Creating and Applying Tablespaces – Filespace Configuration File

```
[gpadmin@mdw:~]$ gpfilespace -o gpfilespace_config  
20131010:15:23:22:003198 gpfilespace:mdw:gpadmin-[INFO]:-  
A tablespace requires a file system location to store its database  
files. A filespace is a collection of file system locations for all components  
in a Greenplum system (primary segment, mirror segment and master instances).  
once a filespace is created, it can be used by one or more tablespaces.  
  
20131010:15:23:22:003198 gpfilespace:mdw:gpadmin-[INFO]:-getting config  
Enter a name for this filespace  
> fs_data1  
  
Checking your configuration:  
Your system has 2 hosts with 1 primary and 1 mirror segments per host.  
Your system has 2 hosts with 0 primary and 0 mirror segments per host.  
  
Configuring hosts: [sdw2, sdw1]  
  
Please specify 1 locations for the primary segments, one per line:  
primary location 1> /data/user_spc/primary  
  
Please specify 1 locations for the mirror segments, one per line:  
mirror location 1> /data/user_spc/mirror  
  
Configuring hosts: [smdw, mdw]  
  
Enter a file system location for the master  
master location> /data/user_spc/master  
20131010:15:23:51:003198 gpfilespace:mdw:gpadmin-[INFO]:-creating configuration file  
20131010:15:23:51:003198 gpfilespace:mdw:gpadmin-[INFO]:-[created]  
20131010:15:23:51:003198 gpfilespace:mdw:gpadmin-[INFO]:-  
To add this filespace to the database please run the command:  
gpfilespace --config /home/gpadmin/gpfilespace_config  
  
[gpadmin@mdw ~]$
```

All directories must exist and be owned by gpadmin

Directories must already exist on segment hosts

Directory must already exist on master and standby hosts

Pivotal.

To create a tablespace, you must first create the underlying filesystem and related filesystem.

Before creating the filesystem:

- Ensure the directory that the filesystem will be associated with already exists on the master, standby, and segment hosts. You can use the gpssh command to create the directory on all hosts in the cluster. For example, the directory on the master and standby servers would be `/data/user_spc/master`. The directory for the primary segments could be `/data/user_spc/primary`, while the mirror segments would use `/data/user_spc/mirror`. If you have multiple primary and mirror segments per segment host, you will need to create separate directories for each of these segments.
- The directories you create must be owned and be writable by the `gpadmin` user.

## **Creating and Applying Tablespaces – Filespace Configuration File (Continued)**

To create the filespace:

1. As a database superuser, use the command, `gpfilespace -o config_file`, to create the configuration file that will eventually be used to create the filespace, where `config_file` is the file to be created.
2. You will be prompted for each primary segment directory and mirror segment directory on all segment hosts. If your cluster hosts multiple primary segments per host, you will need to define a separate directory for each of these primary and their corresponding mirror segments.
3. You will be prompted for the master and standby master directory. This directory is the same on both systems, so this is entered only once.

## Creating and Applying Tablespaces – Creating the Filespace

The screenshot shows a terminal window titled "gpadmin@mdw:~". Inside the window, a configuration file named "gpfilespace\_config" is displayed, containing details about file locations for master, standby, primary, and mirror segments. Below this, a command is run to create a filesystem using this configuration. A callout box highlights the configuration file's purpose: "Configuration file contains all directories needed by masters and segments". Another callout box highlights the command being run: "Create the filesystem as the gpadmin user using the filesystem configuration file". The Pivotal logo is visible in the bottom right corner.

Example: Filespace Configuration File

Configuration file contains all directories needed by masters and segments

```
$ cat gpfilespace_config
filespace:fs_data1
mdw:1:/data/user_spc/master/gpseg-1
smdw:6:/data/user_spc/master/gpseg-1
sdw2:3:/data/user_spc/primary/gpseg1
sdw2:4:/data/user_spc/mirror/gpseg0
sdw1:2:/data/user_spc/primary/gpseg0
sdw1:5:/data/user_spc/mirror/gpseg1
```

gpadmin@mdw:~

```
[gpadmin@mdw ~]$ gpfilespace --config /home/gpadmin/gpfilespace_config
20131010:15:52:18:003805 gpfilespace:mdw:gpadmin-[INFO]:-
A tablespace requires a file system location to store its database
files. A filespace is a collection of file system locations for all components
in a Greenplum system (primary segment, mirror segment and master instances).
once a filespace is created, it can be used by one or more tablespaces.

20131010:15:52:18:003805 gpfilespace:mdw:gpadmin-[INFO]:-getting config
Reading Configuration file: /home/gpadmin/gpfilespace_config
20131010:15:52:18:003805 gpfilespace:mdw:gpadmin-[INFO]:-Performing validation on paths
.....
20131010:15:52:19:003805 gpfilespace:mdw:gpadmin-[INFO]:-Connecting to database
20131010:15:52:19:003805 gpfilespace:mdw:gpadmin-[INFO]:-Filespace "fs_data1" successfully
created
[gpadmin@mdw ~]$
```

Create the filesystem as the gpadmin user using the filesystem configuration file

Pivotal.

Once the filesystem configuration file has been created, you can use it to create the filesystem. The configuration file contains the information you entered during the file configuration step. The directories for the master, standby, primary, and mirror segments are all documented in the file.

To create the filesystem, as the gpadmin user, issue the command,  
`gpfilespace --config config_file,`

where *config\_file* represents the file that contains the filesystem configuration.

## Creating and Applying Tablespaces – Creating the Tablespace

Usage: Syntax to create a tablespace

```
gpadmin=# CREATE TABLESPACE tablespace_name [OWNER username]
FILESPACE filesystem_name
```

gpadmin=# select \* from pg\_filespace;

fsname	fowner
pg_system	10
fs_data1	10

(2 rows)

```
gpadmin=# create tablespace user_s1_tspc filesystem fs_data1;
CREATE TABLESPACE
gpadmin=# select spcname, fsname from pg_tablespace,pg_filespace where pg_filespace.oid=pg_
_tablespace.spcfsid;

```

spcname	fsname
pg_default	pg_system
pg_global	pg_system
user_s1_tspc	fs_data1

(3 rows)

```
gpadmin#
```

Note: The maximum number of tablespaces and filespaces are represented as `gp_max_tablespaces` and `gp_max_filespaces` in the master `postgresql.conf` file.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

Creating the tablespace defines an in-database relationship between the tablespace and the filesystem you created. The tablespace name must be distinct from other tablespaces in the database.

To create the tablespace, enter the following SQL command as a database superuser:

```
CREATE TABLESPACE tablespace_name FILESPACE
filesystem_name
```

As the database superuser, you can assign the tablespace to a non-superuser during the creation process or afterwards with the `ALTER TABLESPACE` command. Note that the tablespace name cannot begin with `gp_` or `pg_` as these prefixes are reserved for system tablespaces.

Note that by default, the maximum number of tablespaces you can define in the Greenplum Database system is 16, while the maximum number of filesystems is 8. These values can be adjusted in the `postgresql.conf` file by changing the following values:

- `gp_max_tablespaces` – Sets the maximum number of tablespaces in the system
- `gp_max_filespaces` – Sets the maximum number of filesystems in the system

The values for these parameters are included in the calculation of space reserved for internal structures for the database. Exceeding these values makes your environment vulnerable to out-of-shared-memory issues. Increasing these values assumes you have enough memory to accommodate the new values.

## Creating and Applying Tablespaces – Applying the Tablespace

Object	Example
Database	<code>CREATE DATABASE tt_db TABLESPACE user_s1_tspc;</code>
Table	<code>CREATE TABLE tt_rt (id int) TABLESPACE user_s1_tspc;</code>
Partitioned Table	<code>CREATE TABLE ttct2_part_rt (id int, id2 int) PARTITION BY LIST (id) (     PARTITION one VALUES (1),     PARTITION two VALUES (2) TABLESPACE user_s1_tspc,     PARTITION three VALUES(3) ) ;</code>
Index	<code>CREATE INDEX tt_idx ON tt_rt (id) TABLESPACE user_s1_tspc;</code>



**Note:** The default tablespace for the environment can be set by modifying the `default_tablespace` parameter in the master server's `postgresql.conf` file.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

11

When creating a database, table, or index, you can specify the tablespace to create the database object in. When you specify the tablespace for a database, all objects created in that database will be created in the same tablespace.

When you do not specify the tablespace on creating an object, the default tablespace for the database is used. You can change the default tablespace for users in Greenplum by modifying the `default_tablespace` parameter in the `postgresql.conf` file on the master server.

## Altering and Removing Tablespaces and Filespaces

Action	Syntax
Changing the tablespace	<code>ALTER TABLESPACE name RENAME TO newname</code> <code>ALTER TABLESPACE name OWNER TO newowner</code>
Removing a tablespace	<code>DROP TABLESPACE [IF EXISTS] tablespacename</code>
Changing a filespace	<code>ALTER FILESPACE name RENAME TO newname</code> <code>ALTER FILESPACE name OWNER TO newowner</code>
Removing a filespace	<code>DROP FILESPACE [IF EXISTS] filespacename</code>



**Note:** You cannot drop a tablespace or filespace until all associated objects are removed from the tablespace and all tablespaces using the specified filespace are removed.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

12

You can change and remove tablespaces if you are the owner, a member of a role that owns the tablespace, or are a database superuser. You can change the name of the tablespace or reassign it to a new owner using the `ALTER TABLESPACE` SQL command. You can remove the tablespace using the `DROP TABLESPACE` SQL command.

The same logic applies to filespaces, where you can change and remove a filespace if you are the owner, a member of a role that owns the filespace, or a database superuser. You can change the name or owner of the filespace using the `ALTER FILESPACE` command and drop the filespace using the `DROP FILESPACE` command.

Before you can drop a filespace, you must remove all tablespaces associated with the filespace. Additionally, if the filespace is being used for temporary or transaction files, it cannot be dropped.

## Additional Supported Table Types



Temporary tables can be used for:

- Storing transient results needed for other session queries
- Perform transformations on data

External tables:

- Facilitate data loading and unloading
- Let you stream data from external sources
- Let you push data out of the database

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

13

Greenplum Database provides support for a variety of tables to facilitate data processing and data loading or unloading. In addition to regular tables, the following table types are supported:

- Temporary tables – These are often used to store the results of queries that are required for a short period of time, during a single connection session, or for transforming data during an ELT process.
- External tables – External tables are used to facilitate data loading and unloading to and from the Greenplum Database. It can be useful in a variety of applications, including to retrieve a stream of data from an external source, such as from a website or operating system command. It can also be used to push data between databases.

External tables have previously been discussed in this course. More details on external table usage and definition will be provided later in the course.

## Temporary Tables – Overview

Temporary tables:

- Are session-specific
- Are dropped at the end of a session
- Take precedence over permanent tables of the same name
- Are created in a special schema created on connection to a session
- Can be distributed, indexed, and analyzed

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

14

Temporary tables in Greenplum Database are session specific, in that each session must create its own temporary table. A temporary table is not shared across sessions and is automatically dropped at the end of the session in which it is created. Any indexes created against the temporary table are also considered temporary and are dropped at the end of the session. As temporary tables are session-specific, you cannot share data across sessions.

If you are not referencing tables by their schema-qualified names, a temporary table takes precedence over existing permanent tables using the same name. Temporary tables are automatically created in a special schema and cannot be assigned to a schema of your choosing. These schemas are automatically created when you connect to a session.

Temporary tables behave like permanent tables in that:

- You can specify a distribution key for a temporary table.
- You can create indexes against columns of a temporary table. These indexes are themselves temporary.
- Temporary tables can be analyzed, allowing the optimizer to generate an execution plan for the table.

They differ from permanent tables in that they are not added to the system catalog and therefore do not impact the size and performance of the system catalog.

## Creating a Temporary Table



### Example: Creating a temporary table

```
gpadmin=# CREATE TEMPORARY TABLE monthlytranssummary (
    storeid      INTEGER,
    customerid   INTEGER,
    transmonth   SMALLINT,
    salesamttot  DECIMAL(10, 2)
)
ON COMMIT PRESERVE ROWS
DISTRIBUTED BY (storeid, customerid);
```

You can define how the temporary table will be handled for transactions with the ON COMMIT clause

The following options to the ON COMMIT clause let you define how a temporary table is handled:

- PRESERVE ROWS – No action is taken on the table
- DELETE ROWS – The table is truncated
- DROP – The table is dropped

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

15

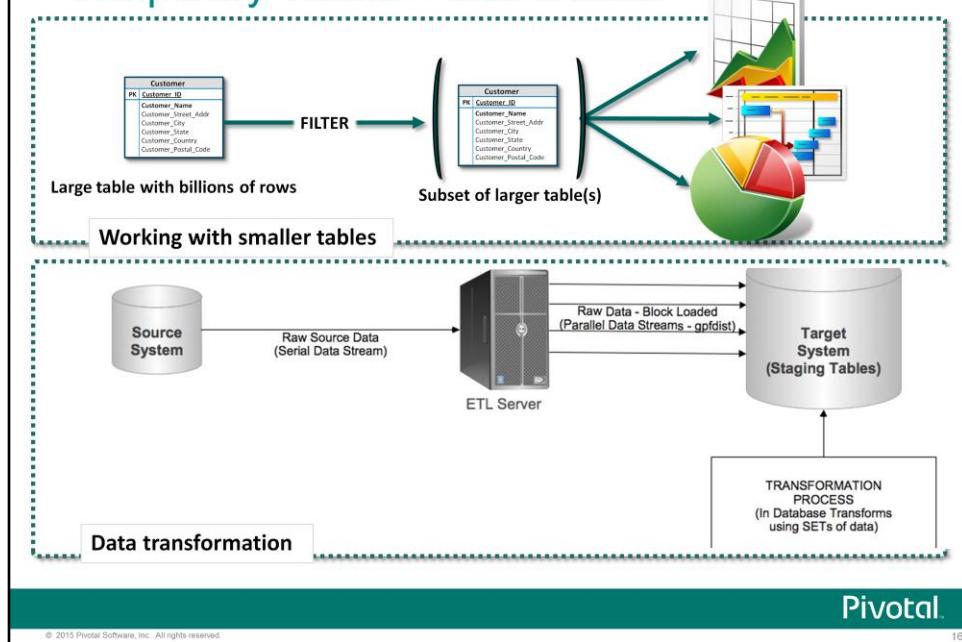
The syntax to create a temporary table is CREATE TEMPORARY TABLE or CREATE TEMP TABLE.

In this example, the temporary table, `monthlytranssummary` is created with the distribution key on two columns: `storeid` and `customerid`. The rows of the temporary table will be preserved, or saved, at the end of the transaction, ensuring that the rows are not removed until the session has ended.

There are three ON COMMIT states that you can pass to the temporary table are:

- PRESERVE ROWS – The rows within the table are saved from removal at the end of a transaction block. This translates into no action being taken on the temporary table or its data.
- DELETE ROWS – The temporary table is truncated at the end of the transaction block, so all rows have been removed.
- DROP – The temporary table is dropped at the end of the transaction block.

## Temporary Tables – Use Cases



The most common use of a temporary table is to take a subset of data from a larger table, reorganize it, and use the results for reporting. The intention is to limit the number of rows you are working with to facilitate reporting. This improves overall performance as you are now working on a subset of data instead of a very large dataset. When working with large tables, it is best to avoid performing a union on a large number of queries against large tables. Instead, if you can reduce the size of your table to just what is needed, you can gain significant performance in generating results. You also reduce the chance of running out of memory when attempting to work with a large amount of rows in memory because the query optimizer will attempt to launch each query in parallel.

Another use case for working with temporary tables is for data transformation. With the extract, load, and transform method, you load data directly into the database and then perform your transformation on the data. By loading the data into the database and performing the transform, you take advantage of the Greenplum MPP architecture to improve performance in an area that is often the slowest part of data migration.

## Temporary Table Use Case – Joining On a Temporary Table

 Example: Extracting data to a temporary table and joining to a large table

```
CREATE TEMPORARY TABLE MAXI AS (
    SELECT COALESCE(MAX(CustomerID),0) AS MaxID
        FROM dimensions.Customer
)
DISTRIBUTED BY (MaxID)
;

SELECT (RANK() OVER (ORDER BY phone)) + MAXI.MaxID,
       custName,
       address,
       city,
       state,
       zipcode,
       zipPlusFour,
       countrycd,
       phone
  FROM public.customer_external
 CROSS JOIN MAXI;
```

Find the maximum customer ID and save it to a temporary table

Iterate over each row in the external table, sorted by phone number, and add a new column that will act as the customer ID, based on adding the maximum value from the temporary table to a ranking value

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

In the example shown, the maximum non-null customer ID is extracted from the table containing customer information. The results are saved to a temporary table, `maxi`. In the next query statement, data is extracted from an external table which contains additional customer information. The information is ordered by the customer's phone number and a rank is assigned to each customer. The rank will increment with each row line. This value is added to the results of the temporary table. The final result is that each customer can be assigned a customer ID starting with the maximum customer ID that is in the `dimensions.customer` table.

## Table Storage Models

**Heap storage**

- Default storage model
- Supports INSERT, UPDATE, DELETE
- Best for:
  - Data that is often modified
  - Smaller dimension tables
- Supports row-oriented tables
- Uses MVCC to support transactions

**Append-optimized storage**

- Append-optimized storage model:
  - Optimized for data warehouses
  - Works best with denormalized data
  - Supports UPDATE and DELETE
  - Best for:
    - Older data
    - Large fact tables
- Supports row and column-oriented tables
- Supports in-database compression
- Uses a Visibility Map (visimap) to hide outdated rows

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

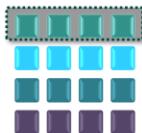
Greenplum provides support for two table storage models with various available options. These storage models used are.

- Heap storage model – Unless otherwise specified, Greenplum tables are created using the heap storage model. Heap tables allow you to add new rows, update existing rows, and remove rows from the table. These types of transactions are seen in OLTP databases where data is often modified after it has been loaded.
- Append-optimized storage model – An append-optimized table is optimized for data warehouses in that it works best with denormalized data. If you follow the strictest definition of the data warehouse, data is not normally modified once it has been loaded to its permanent table. Append-optimized tables afford some flexibility in that the data can be refreshed, allowing updates and deletes. Append-optimized tables work well for large batch uploads instead of single row inserts as would be seen in OLTP-type transactions. Large tables, where data does not normally require refreshing, such as older data, or large fact tables, are ideal use cases for append-optimized tables. Append-optimized storage does not work with transactions that use serializable isolation levels, as would be seen with row-level inserts, and updatable cursors. Append-only storage uses a visibility map to mark the rows which should be visible and those that should not.

Both storage models support row-oriented tables, whereas column-oriented tables are only supported with append-optimized storage. Append-optimized storage also supports in-database compression at the table and columnar levels.

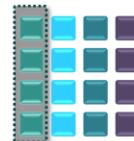
## Row-Oriented and Column-Oriented Tables

Row-oriented storage



- Works well for OLTP workloads
- Supports mixed workloads (INSERTs, UPDATEs, DELETEs, and reads)
- Is supported with both heap and append-optimized storage

Column-oriented storage



- Works with data warehouse workloads
- Works well for data where you aggregate over a small number of columns
- Efficient for data where you modify one column
- Supported on append-optimized storage

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

19

In addition to heap and append-optimized storage models, you can also define your table to be row-oriented or column-oriented.

A row-oriented table meets the needs of OLTP workloads, where you are accessing row and columnar data. It supports mixed workloads where you may perform inserts, updates, deletes, and reads quite frequently.

A column-oriented table, as with append-optimized storage, performs well for data warehouse workloads, where you may oftentimes be computing aggregations of data over a small number of columns. This type of storage also works well for cases where you need to modify very few columns without modifying other columnar data.

## Creating Heap and Append-Optimized Tables

Action	Example
Creating a heap, row-oriented table	<pre>CREATE TABLE tc_heap (id int, descr text) DISTRIBUTED BY (id);</pre>
Creating an append-optimized, row-oriented table	<pre>CREATE TABLE tc_ao (id int, sales float) WITH (appendonly=true) DISTRIBUTED BY (id);</pre>
Creating an append-optimized, column-oriented table	<pre>CREATE TABLE tc_ao_c (id int, sales float) WITH (appendonly=true, orientation=column) DISTRIBUTED BY (id);</pre>



**Note:** You cannot modify the storage or orientation of a table once defined. You can create a new table with the desired options and migrate your data.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

20

Unless otherwise specified, the default behavior when creating a table is to create a heap storage table with row orientation.

To create an append-optimized table, you must include the clause `WITH (appendonly=true)` in your table definition.

To specify column orientation, use the `WITH (appendonly=true, orientation=column)` clause in your table definition.

You cannot change the storage model or orientation of an existing table. You can instead create a new table with the desired storage model and orientation and migrate your data into that table.

## Compressing Table Data

Compression Algorithm	Compression Levels	Description	Table-Level Compression	Row-Level Compression
ZLIB	1 – 9	Offers the most compact ratio with a potential impact to CPU performance	Supported	Supported
QUICKLZ	1	Offers faster, but lower, data compression	Supported	Supported
RLE_TYPE Delta Compression (specific data types)	4	Offers run-length encoding compression for columns based on repeated values	Unsupported	Supported

 **Question:** What type of data do you think would work well with the different offerings of compression?

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

Append-optimized tables support compression at two levels:

- Table-level compression is applied to the entire table with support for ZLIB and QuickLZ compression algorithms.
- Column-level compression is applied to columns within a table. Each column can utilize a different compression algorithm. Supported algorithms at the column-level include RLE\_TYPE, ZLIB, and QuickLZ.

Note that there is an order of precedence. Compression at the subpartition level of a table overrides all other compression defined for the table. The order of precedence from lowest to highest is table, column, partition, and subpartition.

The various algorithms offer different performance and compression levels that may lend itself to some environments and data more than others.

- ZLIB is the default algorithm that offers greater compression, but at lower speeds. At compression level 1, zlib offers the fastest compression with the lowest ration, while at 9, it offers the most compact compression at the slowest speed.
- QuickLZ uses less CPU cycles and compresses data faster at a lower compression ratio than zlib, but offers only one compression level.
- RLE\_TYPE offers run-length encoding (RLE) for column-level compression. With RLE, repeated data is stored as a single value along with a count of the number of times that value is found. As of GPDB 4.3.3, in addition to RLE\_TYPE, Delta compression is also applied for columns that are defined as BITINT, INTEGER, DATE, TIME, or TIMESTAMP.

## **Compressing Table Data**

Compression can be applied to data to preserve space within the cluster. Choosing the appropriate compression algorithm and ratio can impact overall performance in the following ways:

- Data compression can help improve I/O performance by saving disk space. As your disk drives reach capacity, you could potentially see a decrease in performance.
- You can also negatively affect performance on some data. You must weigh the compression ratio against storage savings as compressed data may require more CPU cycles to unpack and retrieve.

## Defining Append-Optimized Compression Tables

Action	Example
Creating a zlib compressed table with compression level 5	<pre>CREATE TABLE tc_ao_zlib5 (id int, sales float) WITH (appendonly=true, compressstype=zlib, compresslevel=5) DISTRIBUTED BY (id);</pre>
Creating a quicklz compressed table	<pre>CREATE TABLE tc_ao_quicklz (id int, sales float) WITH (appendonly=true, compressstype=quicklz) DISTRIBUTED BY (id);</pre>
Creating an AO table with an RLE compressed column and a zlib compressed column	<pre>CREATE TABLE tc_ao_rletype (     id int,     sales float ENCODING (compressstype=zlib,                            compresslevel=3),     salesdate date ENCODING (compressstype=rle_type)) WITH (appendonly=true, orientation=column) DISTRIBUTED BY (id);</pre>

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

23

When applying compression to the table, you include the compression type and level in the WITH clause of your table definition.

In the first example, the zlib compression algorithm is applied to the table with the clause, `WITH (appendonly=true, compressstype=zlib, compresslevel=5)`. The default compression level is 1 if the compresslevel option is not included in the clause.

The second example shows how to define a QuickLZ compressed table by including the `compressstype=quicklz` option in the WITH clause.

In the third example, compression algorithms are applied to two columns. Column compression requires the ENCODING clause as part of the column definition before you can define the storage specification. The `sales` column will use zlib compression at compression level 3, while the `salesdate` column will use rle\_type compression.

Note that when you apply compression, it is best to apply it to the lowest level, where the data resides. If only one column requires compression, apply the compression to the column.

## Defining Default Table Storage Options

gp_default_storage_options		
Options	Level	Command
APPENDONLY	Object level	CREATE TABLE ... WITH (...)
BLOCKSIZE	Role level	ALTER ROLE ... SET ...
CHECKSUM	Database level	ALTER DATABASE ... SET ...
COMPRESSTYPE	System level	gpconfig ...
COMPRESSLEVEL		
ORIENTATION		

A large green downward-pointing arrow is positioned to the right of the table, pointing from the highest priority level (Object level) down to the lowest priority level (System level).



Usage: Update default storage options at role level

```
names=> alter role student set
gp_default_storage_options='appendonly=true,compressstype=zlib';
Names=> set role student;
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

24

The server configuration parameter, `gp_default_storage_options`, lets you define the storage options to automatically apply when creating a table with the `CREATE TABLE` command. Options specified as part of the `CREATE TABLE` command automatically override any settings called for in the parameter.

The options are a comma separated list, consisting of the storage options, `APPENDONLY`, `BLOCKSIZE`, `CHECKSUM`, `COMPRESSTYPE`, `COMPRESSLEVEL`, and `ORIENTATION`.

The default behavior can be set at various levels of the database:

- Role level using the `ALTER ROLE SET ...` command
- Database level with the `ALTER DATABASE SET ...` command.
- Global level using the `gpconfig` command.

Specifying options using the `CREATE TABLE` takes precedence over all other forms. This is done on a one-on-one basis. Next order of precedence is the role level, followed by the database level, and then the system level. The options are not cumulative across the various levels. Only the options specified at each level will be used if not specified at a higher level.

In the example shown, the student account alters the `gp_default_storage_options` parameter with the options, `appendonly=true` and `compressstype=zlib`. After reconnecting to the database as the same account, any tables created by that user will adhere to the default options specified.

## Lab: Table Management

In this lab, you create and manage a variety of Greenplum Database tables.

You will:

- Create a temporary table
- Create tables with compression methods applied
- Create append-optimized tables and apply column-wise orientation to the table

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

25

In this lab, you create a variety of tables supported by Greenplum Database, including temporary, column-wise store, and append-optimized tables. You also apply compression methods to tables you create.

## Module 5: Data Loading and Distribution

### Lesson 1: Summary

During this lesson the following topics were covered:

- Creating and applying tablespaces to database objects
- Various types of supported tables
- Supported table storage models
- Compression methods that can be applied to various tables

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

26

This lesson covered the storage table models supported in Greenplum Database. The various storage models, supported tables, and compression methods are discussed in this model, including how to create the various table types and apply the different compression methods.

## Module 5: Data Loading and Distribution

### Lesson 2: Data Loading

In this lesson, you examine various methods of loading data into Greenplum and examine some performance impacts.

Upon completion of this lesson, you should be able to:

- Load data using external tables and parallel loading utilities
- Describe the `COPY` command
- Identify data loading performance tips

Pivotal

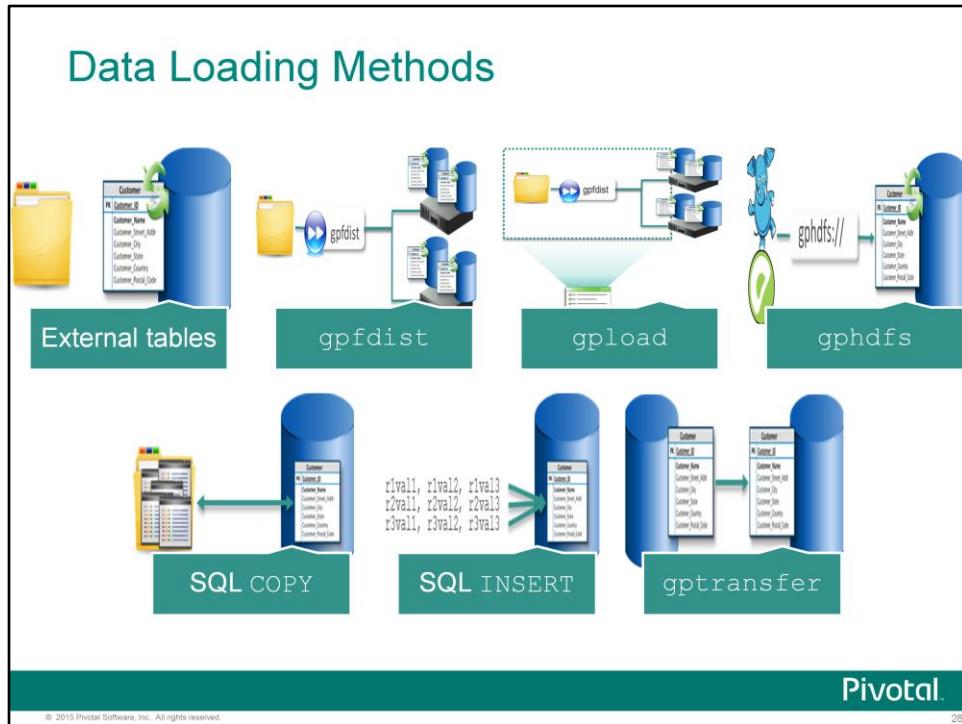
© 2015 Pivotal Software, Inc. All rights reserved.

27

There are various methods of loading data into Greenplum Database. Greenplum supports fast, parallel data loading for large amounts of data. This method can be used for ETL processing.

Upon completion of this lesson, you should be able to:

- Load data into Greenplum using external tables and parallel loading utilities.
- Describe the `COPY` command and how it can be used to load data.
- Identify data loading performance tips that can impact you during and after data loads.



Greenplum supports various methods for loading data into Greenplum Database. These include:

- External tables for data loading and unloading.
- The `gpfdist` utility that is used in conjunction with external tables to provide parallel loads and unloads.
- The `gupload` data loading utility which acts as an interface to the external table parallel loading feature.
- The `gphdfs` protocol acts as an interface between Greenplum Database and the Hadoop filesystem.
- `COPY` command to load or unload data in a non-parallel manner.
- The use of the `SQL INSERT` command to insert data into tables. This method should not be considered for bulk loads of large data volumes. It can be used with external tables to load data into Greenplum Database tables. This option will be discussed later in the course.
- Migrating data from one Greenplum Database system or database to another system or database using the `gptransfer` utility.

## Loading with External Tables

Read-only external tables:

- Leverage parallel processing power of segments
- Can be accessed with `SELECT` statements
- Access data outside of the Greenplum Database
- Commonly used for ETL and data loading

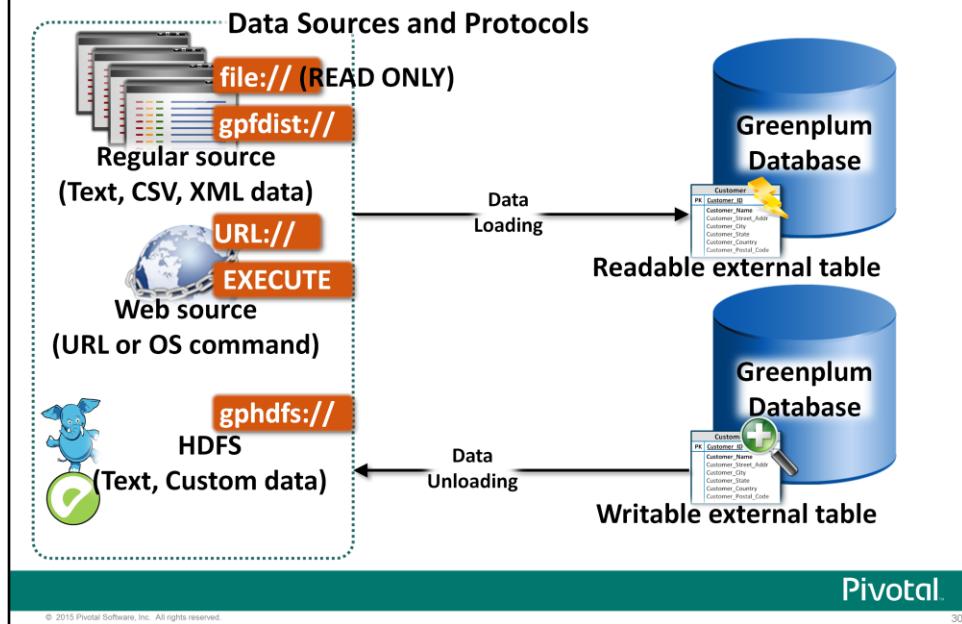
Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

29

External tables leverage the parallel processing power of the segments for data loading. Unlike other loading mechanisms, you can access multiple data sources with one `SELECT` of an external table. The data resides outside of the Greenplum Database. There is a lot of flexibility when defining an external table as to where, how, and what data will be used. External tables are commonly used for ETL and data loading into the Greenplum Database.

## External Table Types



Regular external tables:

- Access static data and is rescannable.
- Uses the `file://` or `gpfdist://` protocols. `gpfdist` is a file server program that serves files in parallel.

Web tables use the `http://` protocol or `EXECUTE` clause to execute an operating system command or script. Data is assumed to be dynamic, meaning that query plans involving web tables do not allow rescanning as the data could change during the course of query execution. This can yield slower query plans as the data must be materialized to the I/O if it cannot fit in memory.

Hadoop-based data access uses Hadoop file systems using the `gphdfs` protocol. Data can be text or a user-defined format.

There is one exception to data unloading: you cannot use the `file://` protocol to perform this action. Any of the others are acceptable.

## File-Based External Tables

When creating file-based external tables:

- Specify up to as many URIs as you have segments in the `LOCATION` clause
- Each URI points to an external data file or data source
- URIs do not need to exist prior to defining the external table
- The URI must exist when the data is queried

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

31

File-based external tables allow you to access flat files as if they were database tables.

When defining file-based external tables:

- You can specify multiple external data sources or URIs (uniform resource identifiers) with the `LOCATION` clause, up to the number of primary segment instances in your Greenplum Database array.
- Each URI points to an external data file or data source.
- The URIs do not need to exist prior to defining an external table.
- The `CREATE EXTERNAL TABLE` command does not validate the URIs specified. You will get an error if the URI cannot be found when querying the external table.

## File-Based External Table Protocol and Format



### Example: Create an external table with multiple URIs

```
CREATE EXTERNAL TABLE ext_expenses (name text, date date,  
amount float4, category text, description text)  
LOCATION (  
'file:///segghost1/dbfast/external/expenses1.csv',  
'file:///segghost1/dbfast/external/expenses2.csv',  
'file:///segghost2/dbfast/external/expenses3.csv',  
'file:///segghost2/dbfast/external/expenses4.csv',  
'file:///segghost3/dbfast/external/expenses5.csv',  
'file:///segghost3/dbfast/external/expenses6.csv',  
)  
FORMAT 'CSV' ( HEADER );
```

Protocol can be file:///  
gpfdist:///  
gphdfs://, or custom

Format can be  
CSV, TEXT, XML,  
or custom

You can define as  
many URLs as you  
have segments

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

32

You may use only one of the following protocols for each external table you define:

- **file://** — If using the `file://` protocol, the external data file or files must reside on a segment host in a location accessible by the Greenplum super user. You can specify up to the number of available segments in your Greenplum Database system. If you have a Greenplum Database system with 8 primary segments and you specify 2 external files, only 2 of the 8 segments will access the external table in parallel at query time. The number of external files per segment host cannot exceed the number of primary segment instances on that host. The host name used in the URI must match the segment host name as registered in the `gp_configuration` system catalog table.
- **gpfdist://** — This protocol retrieves the files using the Greenplum file distribution program.  
**NOTE:** If you use CSV formatted files with the `gpfdist` protocol and your external data files have a header row, do not declare `HEADER` when creating the external table definition. Instead, use the `-h` option when you start the `gpfdist` program.

The `FORMAT` clause:

- Is used to describe how the external table files are formatted.
- Accepts plain text (TEXT) or comma separated values (CSV) format.

If the data in the file does not use the default column delimiter, such as the escape character or null string, you must specify the additional formatting options. Formatting options are the same as the `COPY` command.

## Parallel File Distribution Program

The parallel file distribution program, gpfldist:

- Is a C program that uses HTTP
- Can be run on an external server
- Distributes data at 200 MB/s per gpfldist
- Is configured with the gp\_external\_max\_segs parameter
- Provides full parallelism for best performance

The data load utility, gload:

- Interfaces and invokes gpfldist
- Creates an external table definition
- Executes INSERT, UPDATE, or MERGE to load data

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

33

The parallel file distribution program, gpfldist:

- Is a C program which uses the HTTP protocol.
- Can be run on a server outside the Greenplum array.
- Distributes data at a rate of 200 MB/s per gpfldist executing on the system.
- Can be configured with the parameter: gp\_external\_max\_segs.
- Provides full parallelism for the best performance.

gload is a data loading utility that acts as an interface to Greenplum's external table parallel loading feature. Using a load specification defined in a YAML formatted control file, gload executes a load by:

- Invoking the Greenplum parallel file server program.
- Creating an external table definition based on the source data defined.
- Executing an INSERT, UPDATE or MERGE operation to load the source data into the target table in the database.

## Parallel File Distribution Program Example

```
gpfldist -d /var/load_files/expenses1 -p 8081 >> gpfldist.log  
2>&1 &  
gpfldist -d /var/load_files/expenses2 -p 8082 >> gpfldist.log  
2>&1 &
```

### Example: Creating an external table using the gpfldist protocol

```
CREATE EXTERNAL TABLE ext_expenses  
(name text, date date, amount float4, description text)  
LOCATION (  
    'gpfldist://etlhost:8081/*',  
    'gpfldist://etlhost:8082/*')  
FORMAT 'TEXT' (DELIMITER '|')  
ENCODING 'UTF-8'  
LOG ERRORS INTO ext_expenses_loaderrors  
SEGMENT REJECT LIMIT 10000 ROWS ;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

34

In the example shown, the parallel file distribution program is started and points to the directory, /var/load\_files/expenses. Each instance opens a unique port number: port 8081 in the first example and port 8082 in the second.

The external table that is created connects to the parallel file distribution programs started on ports 8081 and 8082 of the server.

Issuing a `SELECT` statement on the external table allows you to view the data pulled in through the parallel file distribution program into the external table.

## Parallel File Distribution Program Example (Cont)



Example: Load data into a regular table

```
INSERT INTO expenses (SELECT * FROM ext_expenses);
```

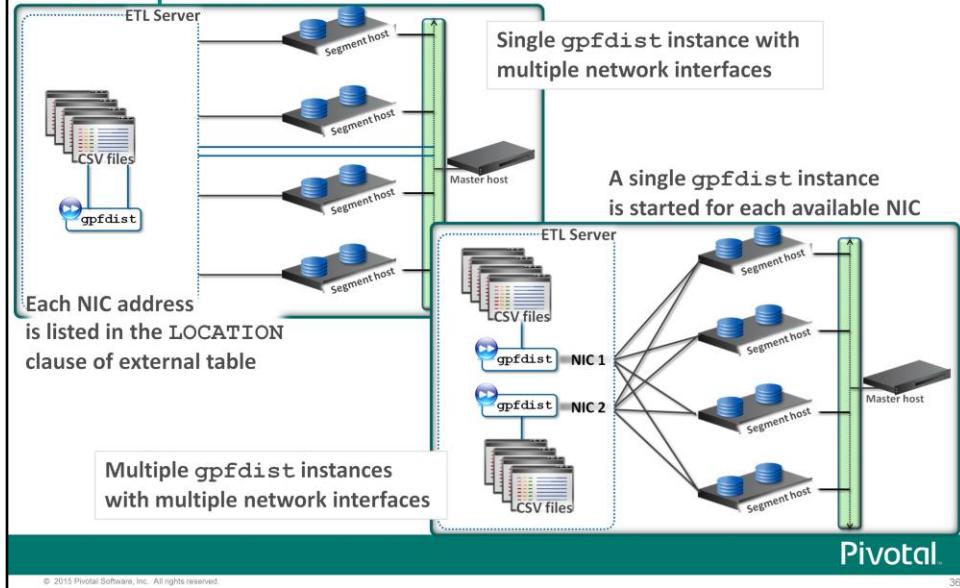
Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

35

Once the external table has been created, data can now be loaded into a regular table. The table has the same columnar definitions as the external table. Once the data has been loaded, it returns how many rows have returned an error and the final number of rows that have been loaded into the table, `expenses`.

## Maximizing `gpfdist` Performance with Multiple NICs



The parallel file distribution program, `gpfdist`, can take advantage of multiple NICs, maximizing network bandwidth between the systems serving the source files, ETL servers, and the Greenplum Database.

In the first example, a single `gpfdist` process is executed on the ETL server. The external table defines two host locations, one for each network interface card in the system. The source files can exist in a single directory on the ETL server. All segment hosts will use all NICs on the ETL simultaneously, thereby increasing bandwidth and helping to improve ingest performance.

The second example shows multiple `gpfdist` instances with multiple network interfaces. To improve the load performance, you divide the data source files evenly among the multiple `gpfdist` programs.

In general, the following steps occur during runtime:

- All segments access all `gpfdist` programs in parallel according to what is defined in the external table definition.
- The segment hosts must be able to connect to the `gpfdist` host using the HTTP protocol to access the files.
- `gpfdist` parses out the data in the files to the segment hosts evenly.

## Loading Data with `gupload`

The data loading utility, `gupload`:

- Acts as an interface to the Greenplum external table parallel loading feature
- Invokes the Greenplum parallel file server, `gpfdist`
- Executes an `INSERT`, `UPDATE` or `MERGE` operation into the target table from the temporary external table
- Creates and drops the temporary external table
- Cleans up `gpfdist` processes
- Does not create intermediate flat files
- Supports pre- and post-SQL statements
- Loads options from a formatted control file

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

37

The `gupload` utility:

- Acts as an interface to the Greenplum external table parallel loading feature, automatically defining the external table in the process.
- Invokes the parallel file server, `gpfdist` on remote server.
- Executes SQL against the external table, including `INSERT`, `UPDATE`, or `MERGE` to load the source data into the target table in the database.
- Drops the temporary external table once it has completed the load.
- Terminates the `gpfdist` processes on the remote server.

The `gupload` utility is a wrapper utility that uses information in a control file to eliminate the manual steps related to bulk loading in parallel. It does not create any intermediate files while processing the data load.

Additional load functionality can be realized by the execution of SQL prior and subsequent to the actual load process as the `gupload` utility supports the use of pre and post SQL statements.

The control file is in a human readable data serialization format, YAML, Yet Another Multicolumn Layout. This borrows concepts from C, Perl and Python and XML.

gpload Control File Format	
Section	Format
Greenplum Database connection information	<pre>DATABASE: db_name USER: db_username HOST: master_hostname PORT: master_port</pre>
gpload definition block	<pre>GPLOAD:   INPUT:     - SOURCE:       LOCAL_HOSTNAME:         - hostname_or_ip       PORT: http_port   PORT_RANGE:         [starting_port,ending_port]     FILE:       - /path/to/input_file   COLUMNS:     - field_name: data_type   FORMAT: text   csv   DELIMITER: 'delimiter_character'   ESCAPE: 'escape_character'   'OFF'   QUOTE: 'csv_quote_character'   HEADER: true   false   ERROR_LIMIT: integer   ERROR_TABLE: schema.table_name</pre>

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

38

### The gpload control file:

- Is based on YAML 1.1
- Implements its own schema for defining the various steps of a Greenplum load operation.
- Specifies the:
  - Greenplum Database connection information.
  - gpfdist configuration information which contains information about the location of the data, the column names and types of the external data table to be created, the expected format, any delimiters and quotes, whether or not a header exists for each data file loaded, the error limit, and the error table to be generated for each error.
  - Destination table that the data will be loaded into. With gpload, the data is automatically loaded after the external table is created. The destination table must therefore exist prior to performing the gpload.
  - Preloading information determines what to do with the table before the data is loaded and whether or not to reuse the external table created as a part of this transaction.
  - Pre and post-SQL statements allow you to specify SQL statements to be executed before and after the data load has occurred.

## gupload Control File Format (Cont)

Section	Format
Destination table information	<pre>OUTPUT:   - TABLE: schema.table_name   - MODE: insert   update   merge   - MATCH_COLUMNS:     - target_column_name   - UPDATE_COLUMNS:     - target_column_name   - UPDATE_CONDITION: 'boolean_condition'   - MAPPING:     target_column_name: source_column_name   'expression'</pre>
Preloading information	<pre>PRELOAD:   - TRUNCATE: true   false   - REUSE_TABLES: true   false</pre>
Pre and Post SQL Statements	<pre>SQL:   - BEFORE: "sql_command"   - AFTER: "sql_command"</pre>

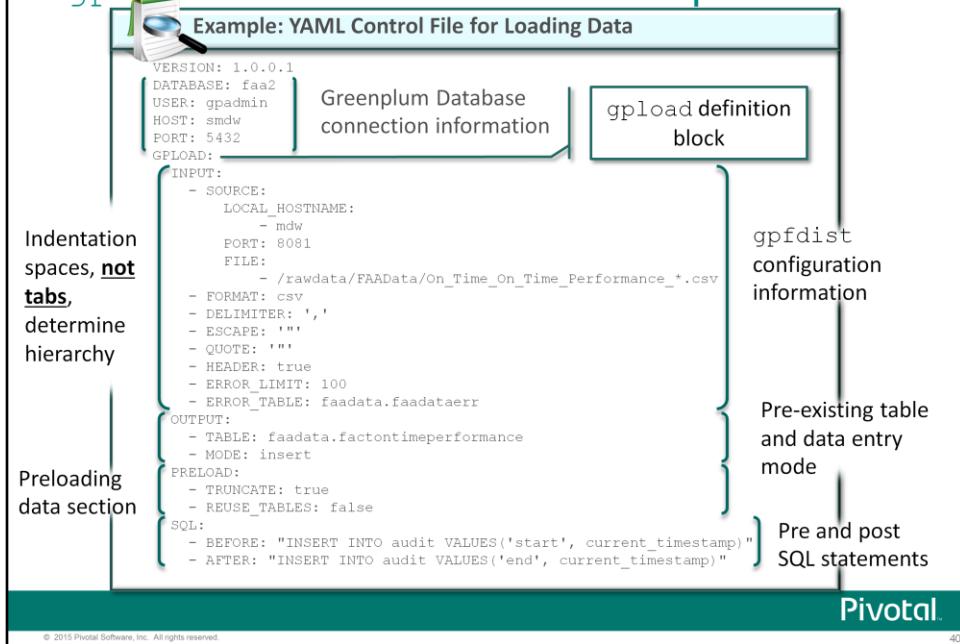
Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

39

The gupload program processes the control file document in order and uses indentation (spaces) to determine the document hierarchy and the relationships of the sections to one another. The use of white space is significant. White space should not be used simply for formatting purposes, and tabs should not be used at all.

## gpload YAML Control File Example



In this example, the YAML control file, `load_faadata.yaml`, defines how data from several source files are to be loaded into the `faa2` database. It is assumed that the `faadata.factontimeperformance` and `public.audit` tables already exist within the database. A reject limit of 100 errors is defined for this load, with any errors redirected to the `faadata.faadataerr` table.

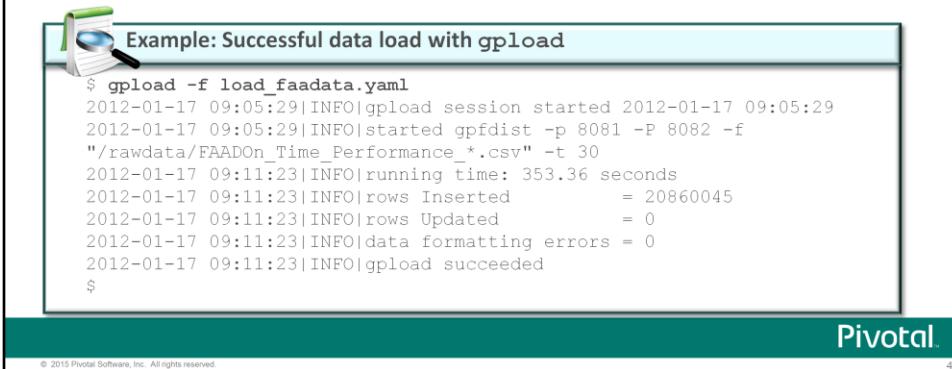
Before the load occurs, the table is truncated. The external table created as a result of this load will not be reused. Reusing a table is useful if you are performing trickle loads or need to reduce system catalog bloat, which can occur when creating and dropping multiple temporary tables. By reusing the table, the table is not destroyed and recreated.

## gupload Syntax

The gupload syntax is as follows:

```
gupload -f control_file [-l log_file] [-h
hostname] [-p port] [-U username] [-d
database] [-W] [-v | -V] [-q] [-D] gupload
-? | --version
```

The following is an example of how it is used:



Example: Successful data load with gupload

```
$ gupload -f load_faadata.yaml
2012-01-17 09:05:29|INFO|gupload session started 2012-01-17 09:05:29
2012-01-17 09:05:29|INFO|started gpfdist -p 8081 -P 8082 -f
"/rawdata/FAADOn_Time_Performance_*.csv" -t 30
2012-01-17 09:11:23|INFO|running time: 353.36 seconds
2012-01-17 09:11:23|INFO|rows Inserted      = 20860045
2012-01-17 09:11:23|INFO|rows Updated       = 0
2012-01-17 09:11:23|INFO|data formatting errors = 0
2012-01-17 09:11:23|INFO|gupload succeeded
$
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

The gupload utility is executed by the superuser with the following syntax:

```
gupload -f control_name
```

The command will display the gpfdist command that was issued while performing the data load.

Once the external table has been created, data is automatically loaded into the target database table specified in the control file. gupload will return a list of the number of rows inserted, updated, as well as any errors detected.

The gupload utility will connect to the database with the user specified with the -U option and the password specified with the -W option.

If the user is not specified on the command line, gupload will extract the user from one of the following in the order shown: the load control file, the environment variable \$PGUSER, or lastly, the current system user name.

As with the user, if the password is not specified on the command line, gupload will extract the password from the following in the order specified: \$PGPASSWORD variable, the password file specified by \$PGPASSFILE or .pgpass if not set.

Finally, if not found in any of these locations, you will be prompted for the password.

Once the control file has been created, issuing the `gupload` command starts a chain of events:

- The `gpfldist` program is started using one of the ports specified in the control file.
  - `gupload` creates the external table based on the information provided in the control file.
  - The data is loaded using the `INSERT INTO` SQL syntax into the target table.

## External Web Table Protocols and Format



### Example: External WEB table with data loads from URLs

```
CREATE EXTERNAL WEB TABLE ext_expenses (name text, date  
date, amount float4, description text)  
LOCATION (  
'http://intranet.company.com/expenses/sales/expenses.csv',  
'http://intranet.company.com/expenses/finance/expenses.csv',  
'http://intranet.company.com/expenses/ops/expenses.csv')  
FORMAT 'CSV' ( HEADER );
```



### Example: External WEB table with data loads from a script

```
CREATE EXTERNAL WEB TABLE log_output (linenum int, message  
text)  
EXECUTE '/var/load_scripts/get_log_data.sh'  
ON HOST FORMAT 'TEXT' (DELIMITER '|');
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

43

In the first example shown on the slide, an external web table is defined using the HTTP protocol. Three different URLs are associated with the external table, all using the CSV format.

The second example shows how to define an external table to a script. The script:

- Must be located on all segments in the same location.
- Must be executable by the superuser account.
- Will execute by one segment on each segment host.
- Executes in parallel on all segments.

## External Web Table Protocols and Format (Cont)



Example: External WEB table with data loads from a command

```
CREATE EXTERNAL WEB TABLE du_space (storage text)
EXECUTE 'du -sh' ON ALL FORMAT 'TEXT';
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

44

In the third example, the external table is created on the command, du -sh. The command executes on all segments on all segment hosts.

## Environment Variables for Command-Based Web Tables

Environment variables accessed from external tables:

- Are not sourced at the segment host

Can be set in the `EXECUTE` clause as follows:

Example: External WEB table with environment variables

```
CREATE EXTERNAL WEB TABLE TEST(segname text, abbrev text)
  EXECUTE 'export HNAME="$HOSTNAME"-SR; echo
    "$HOSTNAME,$HNAME"' FORMAT 'TEXT' (DELIMITER ',');
SELECT * FROM test;
segname | abbrev
-----+-----
sdw2   | sdw2-SR
sdw1   | sdw1-SR
(2 rows)
```

 **Note:** You can disable the use of the `EXECUTE` command in web table definitions by setting `gp_external_enable_exec` to `off`.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

45

When an external table command is executed, that command is executed from within the database and not from a login shell. If you rely on environment variables in external web table commands, such as the `$PATH` variable, keep in mind that the `.bashrc` or `.profile` of the current user will not be sourced at the segment host. You can set desired environment variables from within the `EXECUTE` clause of your external web table definition as follows:

```
CREATE EXTERNAL WEB TABLE ext_table (...) EXECUTE 'export
PATH=/usr/sbin'
```

Note that you can disable the use of the `EXECUTE` command in web table definitions by setting the parameter, `gp_external_enable_exec`, to `off` in the `postgresql.conf` file.

## External Table EXECUTE Variables

Variable	Description
\$GP_CID	Command count of the session executing the external table statement.
\$GP_DATABASE	The database that the external table definition resides in.
\$GP_DATE	The date the external table command was executed.
\$GP_MASTER_HOST	The host name of the Greenplum master host from which the external table statement was dispatched.
\$GP_MASTER_PORT	The port number of the Greenplum master instance from which the external table statement was dispatched.
\$GP_SEG_DATADIR	The location of the data directory of the segment instance executing the external table command.
\$GP_SEG_PG_CONF	The location of the <code>postgresql.conf</code> file of the segment instance executing the external command.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

46

The Greenplum Database variables shown in the table are available to operating system commands executed by an external table. These variables are set as environment variables in the shell that executes the external table command(s). They can be used to identify a set of requests made by an external table statement, at runtime, across the Greenplum Database array of hosts and segment instances.

## External Table EXECUTE Variables (Cont)

Variable	Description
\$GP_SEG_PORT	The port number of the segment instance executing the external table command.
\$GP_SEGMENT_COUNT	The total number of primary segment instances in the Greenplum Database system.
\$GP_SEGMENT_ID	The ID number of the segment instance executing the external table command.
\$GP_SESSION_ID	The database session identifier number associated with the external table statement.
\$GP_SN	Serial number of the external table scan node in the query plan of the external table statement.
\$GP_TIME	The time the external table command was executed.
\$GP_USER	The database user executing the external table statement.
\$GP_XID	The transaction ID of the external table statement.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

47

## External Table EXECUTE Variables (Continued)

## Data Loading with User-Defined Formats

To use a custom format:

- Build the input and output functions as a shared library (for example, C library function fixed-width formatter)
- Create a Greenplum Database function to return a record with the custom format
- Create the external table with the `FORMAT 'CUSTOM' (formatter=function_name, [arg1=xx,...] )` clause

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

48

In addition to supporting data loads with the parallel file distribution program, Greenplum allows users to define their own custom formats, protocols, and locations to be used for data loads. The `FORMAT` clause with the '`CUSTOM`' argument allows you to specify the format of the data. The library that contains the formatting code can be registered as a function that is used for both reading and writing data to the final destination.

The overall purpose of the user-defined formats and protocols is to remove the need for the interim process, `gpfdist`, from the data load when performing an integration project. This method allows you to interact directly with the data in the method you choose, making the communication more streamlined and efficient for the project.

To use the custom format:

- First, build and install the shared library in a Greenplum accessible directory, such as `/usr/local/greenplum-db/lib`.
- Create a Greenplum function that will return a record of the custom format you have defined. The function should point to the installed library as well as the function within the shared library.
- Finally, create the external table specifying the Greenplum Database function you have created using the `FORMAT 'CUSTOM' ((formatter='GBDB_function', [arg1=, ...])` clause.

## Data Loading with Fixed Width Formatter Example



Example: Create a function, `fixedwidth_input`, that calls the C function, `fixedwidth_in`

```
CREATE FUNCTION fixedwidth_input() RETURNS record as '$libdir/fixedwidth.so', 'fixedwidth_in'
LANGUAGE C STABLE;
```

**C library function**



Example: Create the external table using the CUSTOM format clause

```
CREATE READABLE EXTERNAL TABLE ext_students (
    name varchar(20), address varchar(30), age int)
location ('file://sdwl/home/gpadmin/testdata.in')
FORMAT 'CUSTOM'
(formatter='fixedwidth_input', name='20', address='
30', age='4');
```

**C Shared library**

**Using the formatter, specify the width of each column**

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

In this example, a fixed width library, available with Greenplum, is used to load data that is in a fixed width format. The library is located in `/usr/local/greenplum-db/lib/postgresql` as `fixedwidth.so`.

Create the function that will call this library. We will cover functions in greater detail later in the course. At this time, note that the function will return a record, or entire row, based on the format specified in the shared C library function, `fixedwidth_in`, from the shared C library, `fixedwidth.so`.

Once the function has been created, create the external table with the custom format. Pass in the name of each of the columns of the table as arguments to the `FORMAT 'CUSTOM'` clause and specify the width of each column.

## Data Loading with Fixed Width Formatter Example (Cont)

The screenshot shows two examples of fixed-width data loading and querying.

**Example: Sample data, file://sdw1/home/gpadmin/testdata.in**

SMITH	Yabba Yabba Lane	0013
DSMITH	Dabba Dabba Lane	0023
ESMITH	Eabba Eabba Lane	0033
FSMITH	Fabba Fabba Lane	0043
GSMITH	Gabba Gabba Lane	0053

Annotations indicate column widths: "20 characters wide" for the first column, "30 characters wide" for the second, and "4 digits wide" for the third.

**Example: Result of SELECT on external table with custom format**

```
gpadmin=# select * from ext_students;
 name | address | age
-----+-----+-----
 SMITH | Yabba Yabba Lane | 13
 DSMITH | Dabba Dabba Lane | 23
 ESMITH | Eabba Eabba Lane | 33
 FSMITH | Fabba Fabba Lane | 43
 GSMITH | Gabba Gabba Lane | 53
(5 rows)
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

With the data source in the specified location, verify that the content is read in without any issues.

The data source contains three columns with the first column 20 characters wide, the second column 30 characters wide, and the third column, 4 characters wide. The format defined in the external table requires that there is a total of 54 characters per record. If the data source does not meet that requirement, the load will fail.

## External Table Error Handling

When handling errors using external tables:

- Good rows are loaded
- Poorly formatted rows are not loaded. Row types include:
  - Rows with missing or extra attributes
  - Rows with attributes of the wrong data type
  - Rows with invalid client encoding sequence
- CONSTRAINTS are not checked
- Use the following error handling clause:

```
[LOG ERRORS INTO error_table] SEGMENT  
REJECT LIMIT count [ROWS | PERCENT] (  
PERCENT based on  
gp_reject_percent_threshold parameter )
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

51

The most common use of external tables is to select data from the table and load the data into regular database tables. This is typically done by issuing a CREATE TABLE AS SELECT or INSERT INTO SELECT command, where the SELECT statement queries external table data.

By default, if the external table data contains an error, the entire command fails and no data is loaded into the target database table. To isolate data errors in external table data while still loading correctly formatted rows:

- You define an external table with a SEGMENT REJECT LIMIT clause.
- Specify the number of error rows acceptable, on a per-segment basis, after which the entire external table operation will be aborted and no rows will be processed or loaded.

Note that the count of error rows is per-segment, not per entire operation. The following conditions then apply:

- If the per-segment reject limit is not reached, all rows that do not contain an error will be processed.
- If the limit is not reached, all good rows will be processed and any error rows discarded.

If you would like to keep error rows for further examination, you can optionally declare an error table using the LOG ERRORS INTO clause. Any rows containing a format error would then be logged to the specified error table.

## **External Table Error Handling (Continued)**

If PERCENT is used, the percentage of rows per segment is calculated based on the parameter `gp_reject_percent_threshold`. The default is 300 rows.

Single row error isolation mode only applies to external data rows with format errors, such as:

- Extra or missing attributes
- Attributes of a wrong data type
- Invalid client encoding sequences.

Constraint errors such as violation of a NOT NULL, CHECK, or UNIQUE constraint will still be handled in *all-or-nothing* input mode. Constraints are set on the target table, not the external table, so constraint errors are not caught until the violation row is inserted into the target table. A constraint violation will still cause the entire load to fail. However it is easy to use SQL commands to filter out constraint errors as a condition of the INSERT command.

## External Table Error Handling Clause Example



Example: Create an external row with single row error isolation

```
CREATE EXTERNAL TABLE ext_customer
  (id int, name text, sponsor text)
  LOCATION ( 'gpfdist://filehost:8081/*.txt' )
  FORMAT 'TEXT' ( DELIMITER '|' NULL ' ')
  LOG ERRORS INTO err_customer SEGMENT REJECT LIMIT 5 ROWS;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

53

In this example, an external table is created with the clause `SEGMENT REJECT LIMIT`. The data will therefore be scanned in single row isolation mode. This is helpful in isolating errors during data loads from an external table. Rows with formatting errors, extra or missing attributes, attributes of the wrong data type, and invalid client encoding sequences are caught and logged to the `err_customer` database table. A limit of 5 rows has been defined for this error handling routine.

## External Tables and Planner Statistics

Query planning of complex queries on external tables is not optimal because:

- Data resides outside the database
- No database statistics exist for external table data
- Data from external tables are not meant for frequent or ad-hoc access

You can manually set rough statistics in `pg_class`:

```
UPDATE pg_class
    SET reltuples=400000, relpages=400
  WHERE relname='myexttable';
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

54

Query planning of complex queries involving external tables is not optimal because:

- There are no statistics to help the query planner choose a good plan.
- The data is not in the database.
- External tables are really not meant to be used in this way.

As a workaround, you can manually edit `pg_class` and set `reltuples`, the number of estimated rows, and `relpages`, the number of 32K size pages, after creating an external table definition. This is the only external table statistics that is currently used by the planner.

By default this is set to accommodate large tables to be safe – on millions of rows and 1000 pages.

Other than in this use case, it is not recommended to update the `pg_class` catalog table. If an incorrect change is made to the table, it can result in problems for the Greenplum Database instance.

## COPY SQL Command

The `COPY` SQL command:

- Is a PostgreSQL command
- Is optimized for loading a large number of rows
- Loads all rows in one command and is not parallel
- Loads data from a file or from standard input
- Supports error handling similar to external tables

The following is an example of the command:



Example: Copy data from `/data/myfile.csv` into the table specified

```
COPY mytable FROM '/data/myfile.csv' WITH CSV HEADER;
```

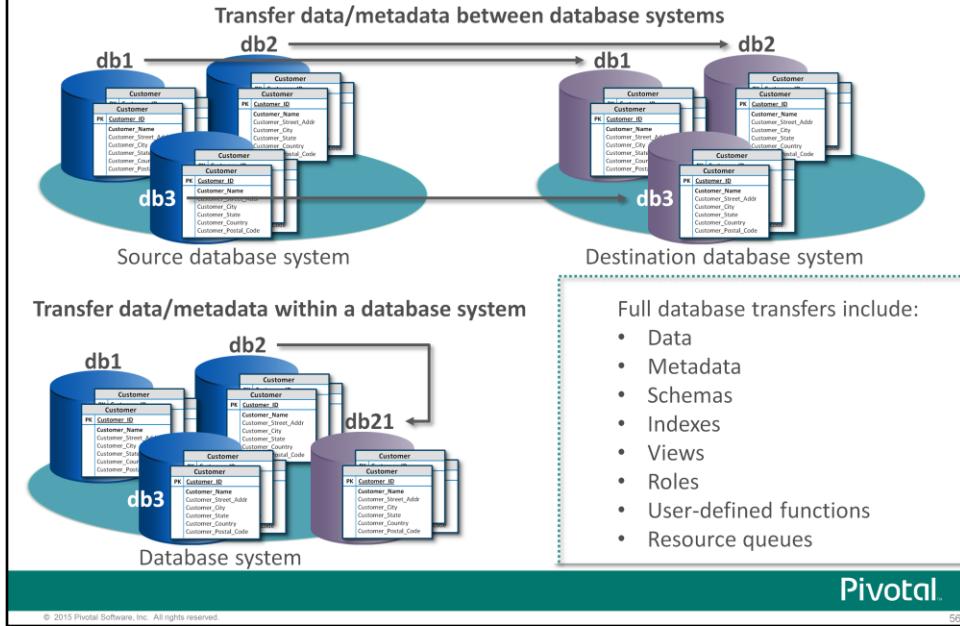
Pivotal  
© 2015 Pivotal Software, Inc. All rights reserved.

The `COPY` command:

- Is a PostgreSQL command that loads multiple rows in one command, instead of using a series of `INSERT` commands.
- Is optimized for loading large numbers of rows.
- Is less flexible than `INSERT`, but incurs significantly less overhead for large data loads. Since `COPY` is a single command, there is no need to disable `autocommit` if you use this method to populate a table.
- It loads data from a file or from standard output.
- Supports error handling.

The table must already exist before copying to the table.

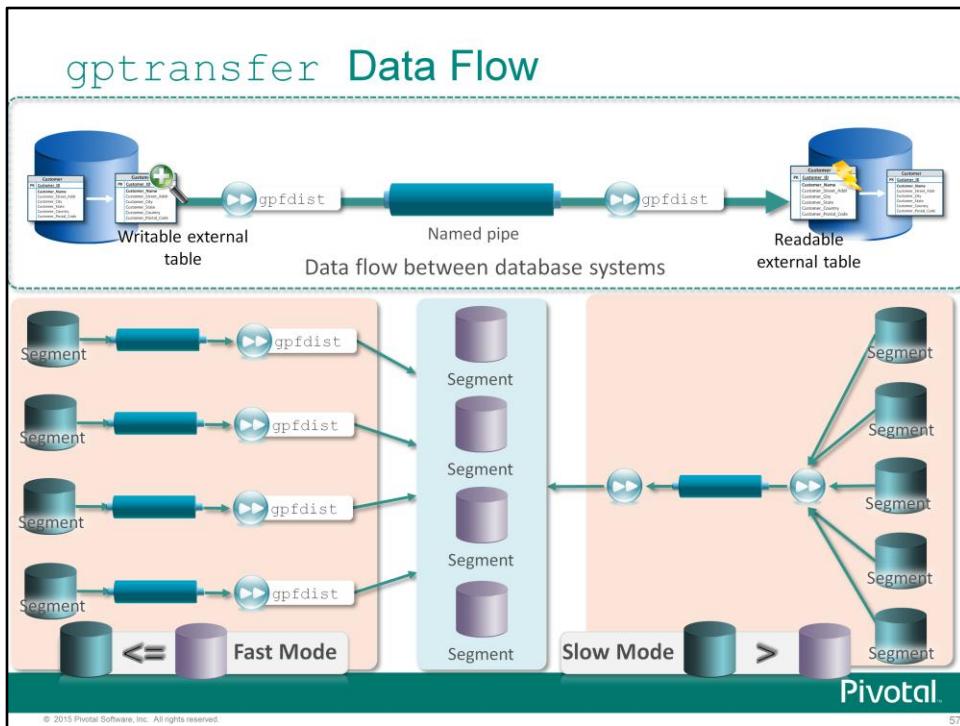
## Migrating Data with gptransfer



The `gptransfer` utility is used to transfer data objects and their metadata between databases. The databases can reside in the same database system or in a different source and target database system. The utility uses `gpfldist` to transfer data in parallel across all segments at the highest transfer rates available to the source and target database systems.

Data can be transferred between two database systems or between databases within the same cluster. The following options for data transfer are available:

- The entire database from one cluster can be transferred to another cluster. All databases, excluding `postgres`, `template0`, and `template1`, are transferred from the host to destination cluster. In addition to transferring the data and metadata for the tables, the database schemas, indexes, views, roles, user-defined functions, and resource queues are also recreated to the destination system. The `postgres.conf` and `pg_hba.conf` files must be manually transferred as these are not included as part of the transfer process. Any extensions that were previously found in the source database system must be installed on the target database system. These are not automatically transferred.
- Database tables that you specify can be transferred from one cluster to another, or within the same cluster to a different target name.

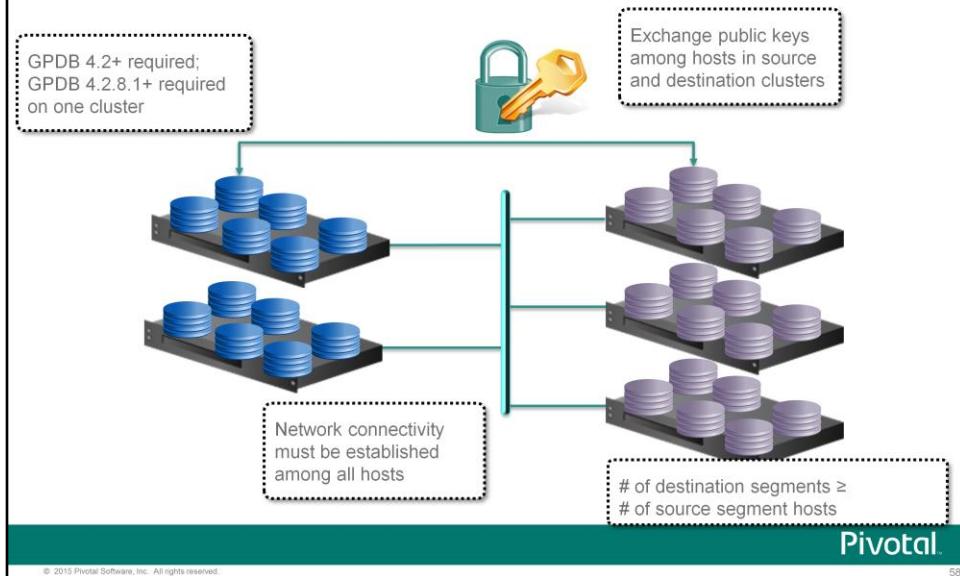


The `gptransfer` utility uses `gpfldist` to transfer data from the source to the destination system. External tables are used for transferring the data, where writable external tables are created on the source database system for each database table to be transferred. A readable external table is created on the destination system for each database table that will be created. Named pipes transfer the data between the writable external table on the source system to the readable external on the target system. Once the transfer is complete, all external tables, named pipes, and `gpfldist` processes are reaped.

The number of `gpfldist` processes used depend on the number of hosts on the source system to the number of segments on the destination system. `gptransfer` uses fast mode or slow mode depending on your setup.

- If the number of segments on the destination cluster is greater than or equal to the number of segments on the source cluster, `gptransfer` uses fast mode. In fast mode, one named pipe and one `gpfldist` process are configured for each segment within the source cluster. Each `gpfldist` process points to a writable external table for that segment. This is an optimal solution that maximizes the data transfer rates between systems.
- If the number of segments in the destination cluster is less than those found in the source cluster, `gptransfer` uses the slow mode. In the slow mode, each segment host is configured with a single writable external table associated a named pipe. Data from all the segments on the segment host are consolidated to the writable external table and are funneled through the `gpfldist` process. The transfer rate is reduced in this case because there are fewer `gpfldist` processes, one per segment host, serving data to the destination cluster.

## Prerequisites for Transferring Data with gptransfer



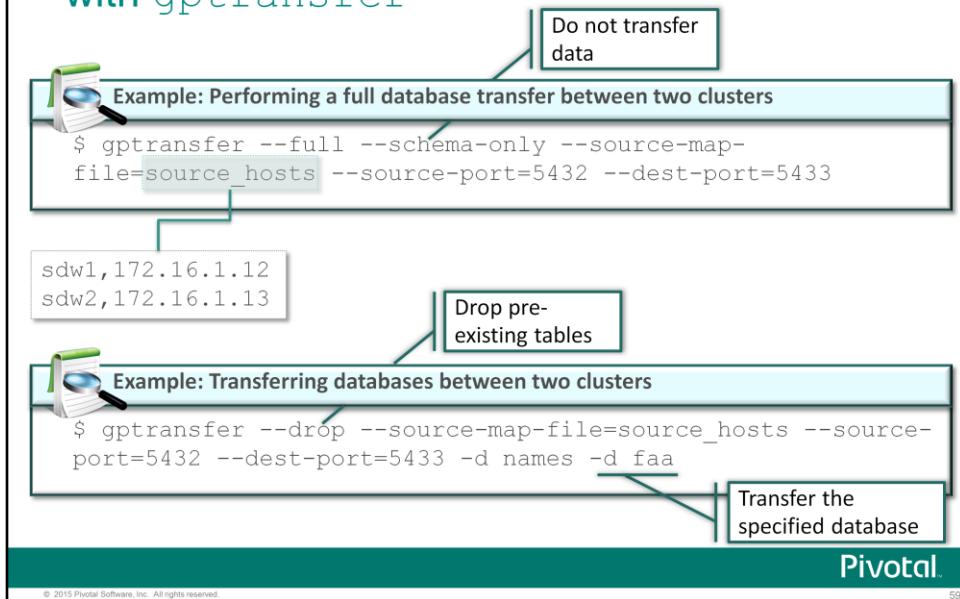
gptransfer requires several conditions are met before attempting to transfer data:

- Data can only be transferred between Greenplum database systems. There is currently no support for other databases, including HAWQ.
- At least one of the Greenplum Database systems must have the `gptransfer` utility installed. The utility was introduced in GPDB 4.2.8.1. This does also require that both systems are at or greater than GPDB 4.2, where one of the systems is running GPDB 4.2.8.1 or higher.
- The utility can be executed from either the source or destination system. If you are transferring data between systems, you must specify which system acts as the source and which acts as the destination system.

There are additional networking and authentication requirements that must be met before proceeding:

- `gptransfer` requires that all segment hosts in both clusters have network connectivity to each other and have exchanged public keys with each other. Create a host file with the list of all segment hosts in both clusters and use the `gpssh_exkeys` utility to exchange public keys among the hosts. `gptransfer` uses IP addresses for communication between clusters to avoid issues with name resolution.
- You must have the same or more segments defined in the destination cluster than there are segment hosts defined in the source cluster. For example, if you have 2 segment hosts in the source cluster, you must have two or more segments in the destination cluster.

## Transferring a Database Between Clusters with gptransfer



The first example highlights the syntax for transferring data from one cluster to another. As the clusters share the same hosts, the port options are used to specify which is the source cluster and which is the destination cluster. When using the `--full` option to transfer between database systems, it is recommended that the data and metadata be transferred in stages. First, transfer and recreate the schema and underlying objects. Once the metadata has been created, transfer the actual data between systems. Use the `--truncate` or `--drop` options to prevent a failure should the table already exist. When performing a full database transfer, the destination database cannot have any preexisting databases other than the system defined databases.

You must also create a host mapping table to define the segment hosts in the cluster along with their IP addresses. Hosts are defined one to a line. If the clusters do not share the same host, include the `--source-host=<hostname>` and `--dest-host=<hostname>` as part of the command line.

The next example shown continues the process of transferring an entire database. You specify the database you wish to transfer using the `-d <database_name>` option. Multiple databases can be specified, each preceded with the `-d` option. In this example, two databases are transferred between systems. The `--drop` option is used to drop the table if it already exists on the destination system.

## Transferring Tables between Databases with gptransfer

Example: Transferring tables within a cluster

```
$ createdb destdb  
$ gptransfer -t names.baby.rank --dest-database destdb  
--truncate
```

Truncate data in a preexisting table

Table to transfer

Destination database

Example: Validate the data transfer of multiple tables

```
$ gptransfer -f table_transfer_list --dest-database destdb  
--skip-existing --validate=md5
```

Ignore the table if it exists on the destination

names.baby.rank  
names.baby.names  
faa.faadata.test\_table

Pivotal

In the first example, a database which will act as the destination for a transfer is created. The table rank, found in the baby schema of the names database is copied to the destination database specified. If the table existed, it is truncated and populated with the data being transferred. Multiple tables can be specified on the command line with each preceded by the `-t` option. As with any data moves, any indexes associated with the table are recreated during the transfer.

The second example shows multiple tables, specified in a file and called with the `-f <filename>` option, are transferred between databases. Each table to be transferred is listed on a line by itself with the notation `database.schema.table`. The `--skip-existing` option specifies that `gptransfer` skip the table if it already exists in the destination database. This transfer is validated using MD5. You can use MD5 or a row count to validate that the data has been transferred successfully. If validation fails, the tables that failed are logged to the file, `failed_migrated_tables_yyyymmdd_hhmmss.txt`, where `yyyymmdd_hhmmss` is a time stamp of when the data transfer process was started.

## Data Loading Performance Tips

Here are several performance tips for data loads:

- Drop indexes and recreate them after loading data
- Increase `maintenance_work_mem` parameter to speed up `CREATE INDEX` operations
- Run `ANALYZE` after load
- Run `VACUUM` after load errors
- Do not use `ODBC INSERT` to load large volumes of data
- Run as many simultaneous `gupload` processes as there are CPU, memory, and network resources available
- Use a DDL script to create empty tables instead of allowing `gptransfer` to transfer them

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

61

The following are performance tips to be aware of when loading data:

- **Drop Indexes Before Loading** — You can load data quickly by creating the table, loading the data, then creating any indexes needed for the table. Creating an index on pre-existing data is faster than updating it as each row is loaded. If you are adding large amounts of data to an existing table, it can be faster to drop the index, load the table, and recreate the index.
- **Increase `maintenance_work_mem`** — Temporarily increasing the `maintenance_work_mem` server configuration parameter speeds up the `CREATE INDEX` commands. It will not help performance of the load itself. Dropping and recreating indexes should be done when there are no users on the system.
- **Always Run `ANALYZE` After Loads** — Whenever you have significantly altered the data in a table, running `ANALYZE` is strongly recommended. Running `ANALYZE` ensures that the query planner has up-to-date statistics about the table. With no statistics or obsolete statistics, the planner may make poor decisions during query planning, leading to poor performance on any tables with inaccurate or nonexistent statistics.

## Data Loading Performance Tips (Continued)

- **Run VACUUM After Load Errors** — Both COPY and selecting from external tables are *all or nothing* operations. Both will stop a load operation if it encounters an error. When this happens, the target table may already have received earlier rows in the or load operation. Although these rows will not be visible or accessible, they still occupy disk space. This may amount to a considerable amount of wasted disk space if the failure happened well into a large load operation. Invoking the VACUUM command will recover the wasted space.
- **ODBC INSERT** — Insert rates, as opposed to COPY and inserting using external tables, can only load data at 60 INSERTs per second when the INSERTs are batched in one big transaction. You can get 400-500 statements per second if the statements are UPDATE, DELETE or SELECT.
- **Run as many simultaneous gload processes as resources** — Take advantage of the CPU, memory, and networking resources available and run as many gload processes at a single time as the system can handle. By taking advantage of resources, you increase the amount of data that can be transferred from ETL servers to the Greenplum Database.
- **Do not use gptransfer to transfer empty tables** — If there are a significant amount of empty tables in your transfer process, consider using a DDL script to recreate those empty tables instead of allowing the utility to do it. The overhead associated with transferring data with gptransfer is significant and should not be used lightly to transfer these tables.

## Lab: Data Loading

In this lab, you load data into the Greenplum instance you have created.

You will:

- 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`

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

63

In this lab, you load data into the Greenplum instance you have created.

## Module 5: Data Loading and Distribution

### Lesson 2: Summary

During this lesson the following topics were covered:

- Loading data using external tables and parallel loading utilities
- COPY command
- Data loading performance tips

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

64

This lesson covered how to load data into new and pre-existing tables using various loading methods, including the parallel loading utilities, external tables, and the COPY command. Performance tips for enhancing and improving data load times was also discussed.

## Module 5: Data Loading and Distribution

### Lesson 3: Table Partitioning

In this lesson, you examine the overall impact of table partitioning and the different table partitioning methods available.

Upon completion of this lesson, you should be able to:

- Describe table partitioning
- Identify the two methods of table partitioning available in Greenplum
- List the reasons for implementing table partitioning
- Identify the steps to partition a table

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

65

Table partitioning allows you to break your tables into consumable pieces that can improve performance during table scans.

Upon completion of this lesson, you should be able to:

- Describe table partitioning in the Greenplum Database.
- Identify the two methods of table available for partitioning a table.
- List the reasons for implementing table partitioning.
- Identify the steps you use to partition a table.

## Partitions in Greenplum

Partitions:

- Are a mechanism in Greenplum for use in physical database design
- Increase the available options to improve the performance of a certain class of queries
- Propagate data to the child tables

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

66

Table partitioning is used to logically divide large tables to improve query performance and facilitate data warehouse maintenance tasks.

All of the schema information from the parent table propagates to its children through table inheritance. However, parent and child need not have the same columns. It is important to keep in mind that parent tables contain no data and indexes and privileges are not passed from parent table to child tables. Therefore, if using indexes explicitly create indexes on each child table and grant permissions on each child table.

## Table Partitioning Overview

Table partitioning:

- Addresses problems in supporting very large tables
- Divides data into smaller, more manageable pieces
- Can improve query performance
- Can be used to facilitate database maintenance tasks
- Uses table inheritance and constraints
- Continues to allow data to be distributed across segments

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

67

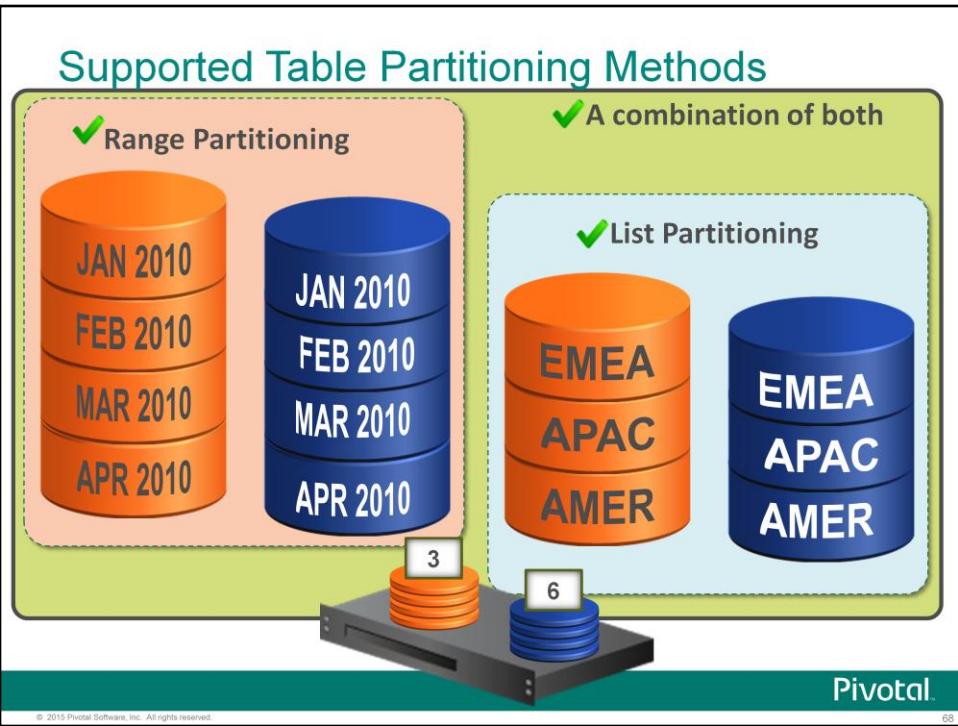
Table partitioning:

- Addresses the problem of supporting very large tables, such as fact tables, by allowing you to divide them into smaller and more manageable pieces.
- Can improve query performance by scanning only the relevant data needed to satisfy a given query. Data is eliminated either during the planning phase or at runtime. This helps to reduce the number of rows returned for joins and further actions.
- Can be used to facilitate database maintenance tasks such as rolling old data out of the data warehouse or speeding up the update of indexes.
- Works using table inheritance and constraints.

Table inheritance creates a persistent relationship between a child table and its parent table(s), so that all of the schema information from the parent table propagates to its children.

CHECK constraints limit the data a table can contain based on some defining criteria. These constraints are also used at runtime to determine which tables to scan in order to satisfy a given query.

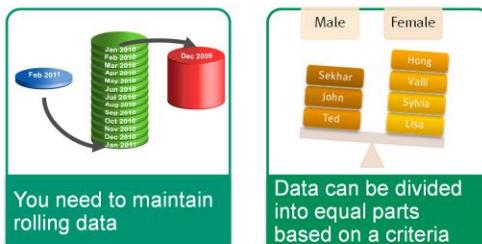
In Greenplum Database, partitioned tables are distributed across the segments as is any non-partitioned table. Partitioning is the method of logically dividing big tables to improve query performance and maintenance. It does not effect the physical distribution of the table data.



Greenplum Database supports:

- Range partitioning, where the data used for partitioning is based on a numerical range, such as date or price.
- List partitioning, where data used for partitioning is based on a list of values, such as gender or region.
- A combination of range and list partitioning.

## When Do You Partition?



Pivotal.

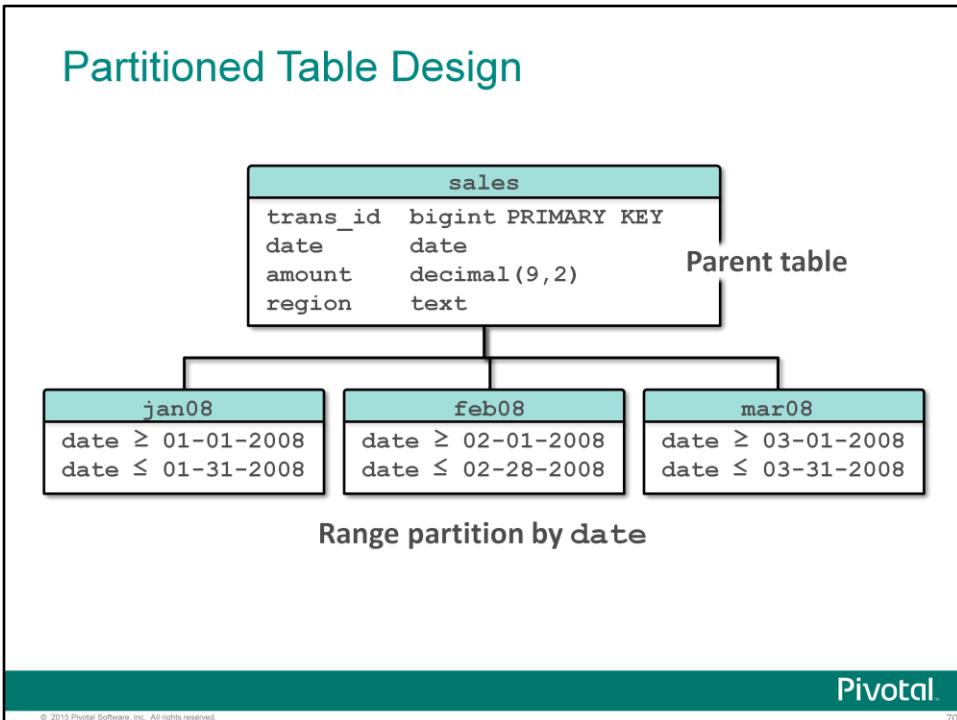
© 2015 Pivotal Software, Inc. All rights reserved.

69

There are several reasons why you would partition your data:

- **Large fact table** – Large fact tables are good candidates for table partitioning. If you have millions or billions of records in a table, you will see performance benefits from logically breaking that data up into smaller chunks. For smaller tables with only a few thousand rows or less, the administrative overhead of maintaining the partitions will outweigh any performance benefits you might see.
- **Unsatisfactory performance** – As with any performance tuning initiative, a table should be partitioned only if queries against that table are producing slower response times than desired.
- **Identifiable access patterns** – Examine the WHERE clauses of your query workload and look for table columns that are consistently used to access data. For example, if most of your queries look up records by date, then a monthly or weekly date-partitioning design might be beneficial.
- **Maintaining rolling data** – Is there a business requirement for maintaining historical data? Your data warehouse may require you keep the past twelve months of data. If the data is partitioned by month, you can drop the oldest monthly partition and load current data into the most recent monthly partition.
- **Dividing data into equal parts** – Choose a partitioning criteria that will divide your data as evenly as possible. If the partitions contain a relatively equal number of records, query performance improves based on the number of partitions created. For example, by dividing a large table into 10 partitions, a query will execute 10 times faster than it would against the unpartitioned table.

## Partitioned Table Design



When you partition a table in Greenplum Database, you:

- Create a top-level or parent-level table
- Create one or more levels of sub-tables, or child tables
- Push the data directly into the lowest level of the table

Internally, Greenplum Database creates an inheritance relationship between the top-level table and its underlying partitions.

Each partition is created with a distinct CHECK constraint. This limits the data that the table can contain. The CHECK constraints are also used by the query planner to determine which table partitions to scan in order to satisfy a given query predicate.

Partition hierarchy information is stored in the Greenplum system catalog so that rows inserted into the top-level parent table appropriately propagate to the child table partitions.

## Creating Partitioned Tables

A table:

- Can be partitioned only at creation time
- Can be partitioned with the command, `CREATE TABLE`
- Can be subpartitioned into additional levels
- Can be partitioned using the following partition designs:
  - Date range
  - Numeric range
  - List

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

71

A table can only be partitioned at creation time using the `CREATE TABLE` command.

The first step in partitioning a table is to decide on the partition design (date range, numeric range, or list of values) and choose the column(s) on which to partition the table. Decide how many levels of partitions you want. For example, you may want to date range partition a table by month and then further subpartition the monthly partitions by sales region.

You can partition a table using:

- A date range
- A numeric range
- A list

## Creating a Ranged Partition Table



### Example: Creating a partition table with date range table partitions

```
CREATE TABLE sales (id int, date date, amt decimal(10,2))
DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
( PARTITION Jan08 START (date '2008-01-01') INCLUSIVE ,
  PARTITION Feb08 START (date '2008-02-01') INCLUSIVE ,
  PARTITION Mar08 START (date '2008-03-01') INCLUSIVE ,
  ...
  PARTITION Dec08 START (date '2008-12-01') INCLUSIVE
  END (date '2009-01-01') EXCLUSIVE );
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

72

You can define a date range or a numeric range for a partitioned table:

- A date range uses a single date or timestamp column as the partition key column. You can use the same partition key column to further subpartition a table if necessary. For example, you can partition by month and then subpartition by day. When date partitioning a table, consider partitioning by the most granular level you are interested in. For example, partition by day and have 365 daily partitions, rather than partition by year then subpartition by month then subpartition by day. A multi-level design can reduce query planning time, but a flat partition design will execute faster at query run time.

You can have Greenplum Database automatically generate partitions by giving a START value, an END value, and an EVERY clause that defines the partition increment value.

By default, START values are always inclusive and END values are always exclusive.

**Note:** You can declare and name each partition individually.

## Creating a Ranged Partition Table (Cont)



### Example: Creating a partition table with numeric range table partitions

```
CREATE TABLE ranking (id int, rank int, year int, gender  
char(1), count int)  
DISTRIBUTED BY (id)  
PARTITION BY RANGE (year)  
( START (2001) END (2008) EVERY (1),  
  DEFAULT PARTITION extra );
```

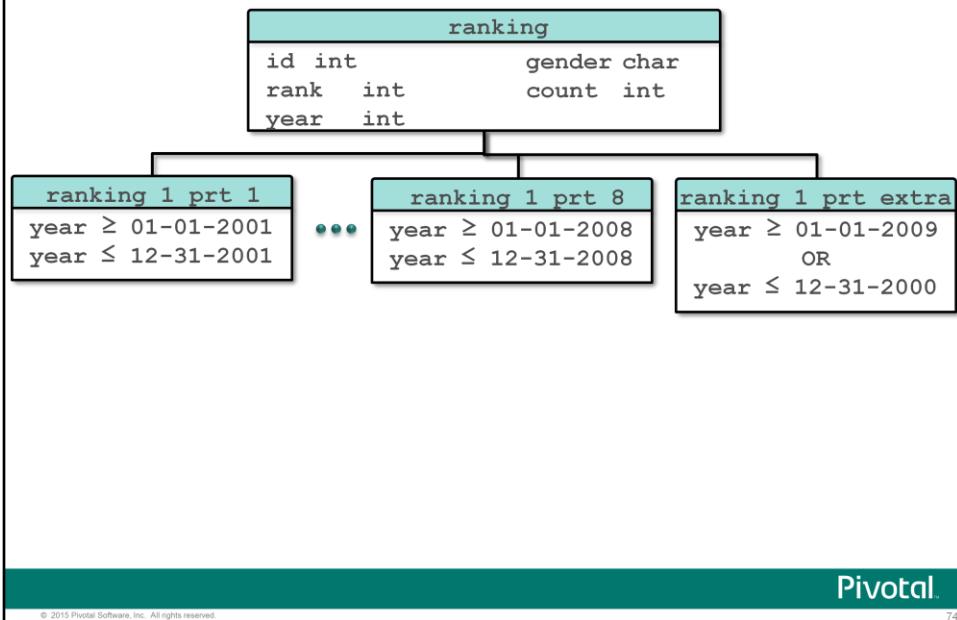
Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

73

- A numeric range uses a single numeric data type column as the partition key column.

## Creating a Ranged Partition Table (Cont)



- The child tables created are created with the name, `ranking_1_prt_#`, where `#` represents the sequential number assigned to the table. In this example, the first child table created for the year 2001 is assigned the name, `ranking_1_prt_1`, while the last child table created is `ranking_1_prt_8`. The default child table created is `ranking_1_prt_extra`.

## Creating a List Partitioned Table



### Example: Creating a partition table with list

```
CREATE TABLE ranking (id int, rank int, year int, gender
char(1), count int )
DISTRIBUTED BY (id)
PARTITION BY LIST (gender)
( PARTITION girls VALUES ('F'),
PARTITION boys VALUES ('M'),
DEFAULT PARTITION other );
```

Pivotal.

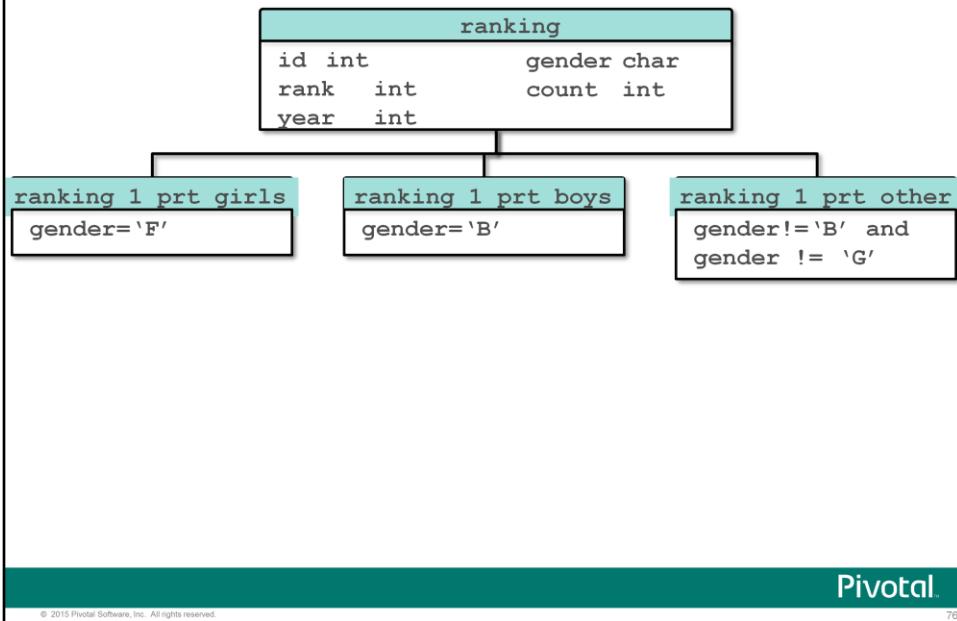
© 2015 Pivotal Software, Inc. All rights reserved.

75

A list partitioned table:

- Can use any data type column that allows equality comparisons as its partition key column.
- Can have a multi-column, or composite, partition key, while a range partition only allows a single column as the partition key.
- Requires you declare a partition specification for every partition, or list value, you want to create.

## Creating a List Partitioned Table (Cont)



For the example provided, each list was assigned a specific name for a specific value. All entries not defined will be assigned to the default table.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

76

## Partitioning an Existing Table

When working with an existing table:

- You cannot partition the data if not already partitioned
- You can create a new partitioned table and migrate your data

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

77

It is not possible to partition a table that has already been created. Tables can only be partitioned at `CREATE TABLE` time.

To partition an existing table, you must:

1. Recreate the table as a partitioned table.
2. Reload the data into the newly partitioned table.
3. Drop the original table and rename the partitioned table to the original name.
4. Grant any table permissions to the new table that were granted on the original table.

## Partitioning an Existing Table Example



### Example: How to partition an existing table

```
CREATE TABLE sales2 (LIKE sales)
PARTITION BY RANGE (date)
( START (date '2008-01-01') INCLUSIVE
END (date '2009-01-01') EXCLUSIVE
EVERY (INTERVAL '1 month') );

INSERT INTO sales2 SELECT * FROM sales;
DROP TABLE sales;
ALTER TABLE sales2 RENAME TO sales;
GRANT ALL PRIVILEGES ON sales TO admin;
GRANT SELECT ON sales TO guest;
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

78

The example shown highlights one method of partitioning an existing table. To perform this action, a second table is created based on the definition of the table to be partitioned. While creating the table, you define the partition for the table, in this case, using a date range.

Once the table has been created, insert data from the previous table into the newly created and partitioned table. The data will be automatically segregated based on the partition you have defined.

Next, drop the original table and change the name of the new table to the name of the original table and grant all appropriate privileges to the table.

## Maintaining Partitioned Tables

Use ALTER TABLE to:



### Example: Add a partition to an existing partitioned table

```
ALTER TABLE sales ADD PARTITION  
START (date '2009-02-01') INCLUSIVE  
END (date '2009-03-01') EXCLUSIVE;
```



### Example: Rename a partition

```
ALTER TABLE sales RENAME TO globalsales;  
ALTER TABLE sales RENAME PARTITION FOR ('2008-01-01') TO  
jan08;
```



### Example: Add a default partition

```
ALTER TABLE sales ADD DEFAULT PARTITION other;
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

79

You maintain a partitioned table using the ALTER TABLE command to:

- Add a partition to an existing partition design. You can:
  - Add a partition to a table that already has subpartitions defined using the ADD PARTITION clause
  - Add a partition to a table that does not have subpartition templates defined using the ADD PARTITION clause followed by the SUBPARTITION statements
  - Add a subpartition using ALTER PARTITION FOR (...) ADD PARTITION ...
- Rename a partition by renaming the top-level parent table to update the child table created or by renaming the partition directly. You cannot directly rename the child table name.
- Add a default partition to an existing partition design. If the partition design is multi-level, you must define a partition default for each level.

## Maintaining Partitioned Tables (Cont)

### Example: Remove a partition

```
ALTER TABLE sales DROP PARTITION FOR (RANK(1));
```

### Example: Truncate a partition

```
ALTER TABLE sales TRUNCATE PARTITION FOR (RANK(1));
```

### Example: Exchange a partition

```
CREATE TABLE jan08 (LIKE sales) WITH (appendonly=true);
INSERT INTO jan08 SELECT * FROM sales_1_prt_1 ;
ALTER TABLE sales EXCHANGE PARTITION FOR ('2008-01-01') WITH
TABLE jan08;
```

### Example: Split an existing partition in a partition table

```
ALTER TABLE sales SPLIT PARTITION FOR ('2008-01-01')
AT ('2008-01-16')
INTO (PARTITION jan081to15, PARTITION jan0816to31);
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

80

- Drop a partition using the `DROP PARTITION` clause. When you drop a partition that has subpartitions, the subpartitions and all data are automatically dropped.
- Truncate a partition using the `TRUNCATE PARTITION` clause. When you truncate a partition that has subpartitions, the subpartitions are automatically truncated.
- Exchange a partition using the `EXCHANGE PARTITION FOR` clause. Exchanging a partition involves swapping in another table in place of an existing partition. You can only exchange partitions at the lowest level of the partition hierarchy. Only partitions that contain data can be exchanged.
- Split a partition using the `INTO` clause. Splitting a partition involves dividing an existing partition into two. You can only split partitions at the lowest level of the partition hierarchy. Only partitions that contain data can be split. The split value you specify goes into the latter partition.

## Viewing Details on Partitioned Tables

**gpadmin@mdw:~**

```
names=# \d+ ranking
Table "baby.ranking"
column | type | modifiers | storage | description
-----+-----+-----+-----+-----
id | integer | | plain | 
rank | integer | | plain | 
year | integer | | plain | 
gender | character(1) | | extended | 
count | integer | | plain | 
Child tables: ranking_1_prt_boys,
ranking_1_prt_girls,
ranking_1_prt_other
Has OIDs: no
Distributed by: (id)
names=#

```

**gpadmin@mdw:~**

```
names=# select partitionboundary, partitiontablename, partitionname, partitiontype, partitionlistvalues
FROM pg_partitions
WHERE tablename=
'ranking';
partitionboundary | partitiontablename | partitionname | partitiontype | partitionlistvalues
-----+-----+-----+-----+-----
PARTITION girls VALUES('F') | ranking_1_prt_girls | girls | list | 'F':>bpchar
PARTITION boys VALUES('M') | ranking_1_prt_boys | Boys | list | 'M':>bpchar
DEFAULT PARTITION other | ranking_1_prt_other | other | list | 
(3 rows)
names=#

```

**gpadmin@mdw:~**

```
names=# select * from pg_partition_columns;
schemaname | tablename | columnname | partitionlevel | position_in_partition_key
-----+-----+-----+-----+-----
baby | ranking | gender | 0 | 1
(1 row)
names=#

```

**Pivotal.**

© 2015 Pivotal Software, Inc. All rights reserved.

**Details on the parent and child tables**

**pg\_partitions view lets you obtain more details on partitioned tables**

**pg\_partition\_columns displays the column used for partitioning**

Some partitioning information can be obtained on tables with the `\d+` PSQL command. All child tables are displayed by their names.

You can obtain additional information by viewing the `pg_partitions` view. In the example shown, the `partitionboundary` column displays how the partitions are divided and the values used to create the specific partition. Next, the partition table name is displayed with the `partitiontablename` column. The partition name follows with the `partitionname` column, followed by the partition rank. The `partitiontype` column displays whether the partition is a list or ranged column. As it is a list column, the `partitionlistvalues` is used to display the values and the data type. The `partitionrangestart` and `partitionrangeend` columns provides similar information for ranged tables.

The `pg_partition_columns` view displays the column used for partitioning. The `position_in_partition_key` column is used to display the position of the column in a multi-column composite partition key found in list partitions.

## Lab: Table Partitioning

In this lab, you create a partitioned table based on a design presented to you.

You will:

- Create and manage table partitions

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

82

In this lab, you create a partitioned table based on a design presented to you. You then modify the table design.

## Module 5: Data Loading and Distribution

### Lesson 3: Summary

During this lesson the following topics were covered:

- Table partitioning
- Methods of table partitioning available in Greenplum
- Reasons for implementing table partitioning
- Steps to partition a table

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

83

This lesson covered how to partition a table, where the data is stored within the table, and the various table partition methods. It also discussed why you would want to partition your table, the performance improvements that you could see with partitioning, and the potential impacts. The steps to create and manage partitions were provided.

## Module 5: Summary

Key points covered in this module:

- Created tablespaces and tables with various storage and compression options
- Loaded data into a Greenplum database instance using external tables, parallel loading utilities, and SQL COPY
- Used table partitioning to logically divide data into smaller parts

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

84

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

- Create and manage tablespaces and tables of varying storage methods
- Load data into a Greenplum database instance using external tables, parallel loading utilities, and SQL COPY
- Use table partitioning to logically divide data into smaller parts.

# Module 6: Database Management and Archiving

This module examines some of the system administrative tasks that you may encounter in the Greenplum environment.

Upon completion of this module, you should be able to:

- Perform system administrative tasks, including managing and checking the state of the Greenplum Database and its data as well as checking the distribution of data
- Perform backup and restoration of Greenplum data from and into the Greenplum Database respectively

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

1

System maintenance is one of the key services that administrators provide for the environment they support. You may be required to make changes to parameters that require restarting the Greenplum environment. Monitoring the system requires knowing not only its current state, but also the state of any applications it supports. Data backup and restoration services are key to reducing the chances of data loss.

In this module, you will:

- Perform system administrative tasks for the Greenplum environment which includes managing and checking the state of the Greenplum Database as well as the state and distribution of data.
- Perform backup and restoration of Greenplum data from the Greenplum Database and back into the Greenplum Database.

# Module 6: Database Management and Archiving

## Lesson 1: Managing the Greenplum Database

In this lesson, you examine the administrative tasks that should be performed to maintain an optimal environment.

Upon completion of this lesson, you should be able to:

- Start and stop the Greenplum Database
- Verify the state of the Greenplum Database
- Use the Greenplum administrative schema to view which tables are exhibiting data skew
- Access log files and logging parameters
- Maintain the system catalog and reclaim physical disk space
- Gather data for troubleshooting issues

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

2

As an administrator, you will encounter times where you must perform specific administrative tasks to keep the Greenplum Database environment running smoothly. You should be aware of how to start, stop, and restart the Greenplum Database and to keep track of the current state of the database. You may also be required to monitor the state of data and log files to maintain an optimal environment.

Upon completion of this lesson, you should be able to:

- Start and stop the Greenplum Database.
- Verify the state of the Greenplum Database using Greenplum commands.
- Use the Greenplum administrative schema to view which tables are exhibiting data skew.
- Access log files and logging parameters.
- Maintain the system catalog and reclaim physical disk space.
- Gather data that can be used for troubleshooting the environment should an issue arise.

## System Administration Tasks – Overview

The following are routine administrative tasks:



Starting and stopping Greenplum



Obtaining the state of Greenplum



Checking for data skew



Accessing log files and parameters



Maintaining the system catalog and reclaiming disk space



Recover Down Segments

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

3

There are several routine tasks that you will perform while maintaining the Greenplum Database. These include:

- **Starting and stopping the Greenplum Database** – Changing a global parameter may necessitate a restart of the Greenplum Database. Because the Greenplum Database is distributed across segments and the master server, it is necessary to understand the shutdown and startup procedures before undertaking this task.
- **Obtaining the state of Greenplum** – A segment failure may go undetected if you do not have notifications configured and mirrors are enabled. Checking the state of the Greenplum environment lets you verify the state of components and processes.
- **Checking for data skew** – Data skew can impact the performance of queries submitted to the Greenplum Database. Using the Greenplum administrative schema, `gp_toolkit`, you can verify how data is distributed across the segments. There are other commands that allow you to achieve that same goal.
- **Accessing log files and parameters** – Log files act as your audit trail for how components within the Greenplum Database are behaving. You must be aware of how to access these log files as part of your monitoring tasks as well as how to rotate them, if necessary.

## **System Administration Tasks – Overview**

- **Maintaining the system catalog and reclaiming disk space** – Numerous database drops and create can cause growth in the size of the system catalog that could affect performance. Regular maintenance helps optimize system performance. If data is constantly changing, vacuuming tables should be on the list of tasks that you regularly perform.
- **Recover Down Segments**– In a system with mirrors enabled, the gprecoverseg utility reactivates a failed segment instance and identifies the changed database files that require resynchronization. Once gprecoverseg completes this process, the system goes into *resynchronizing* mode until the recovered segment is brought up to date. The system is online and fully operational during resynchronization.

## Starting and Stopping Greenplum

The following commands start and stop Greenplum:

Action	Greenplum Application
Start the Greenplum Database	gpstart
Stop the Greenplum Database	gpstop
Restart the Greenplum Database	gpstop -r
Start the Greenplum Database in restricted mode	gpstart -R
Reload master postgresql.conf and pg_hba.conf	gpstop -u
Start the master in utility mode	gpstart -m
Stop the master that was started in utility mode	gpstop -m

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

5

In a Greenplum DBMS, each database server instance must be started or stopped across all of the hosts in the system in such a way that they can all work together as a unified DBMS. This is because the Greenplum Database system is distributed across many machines and database instances.

Greenplum provides the following commands to change the state of the database:

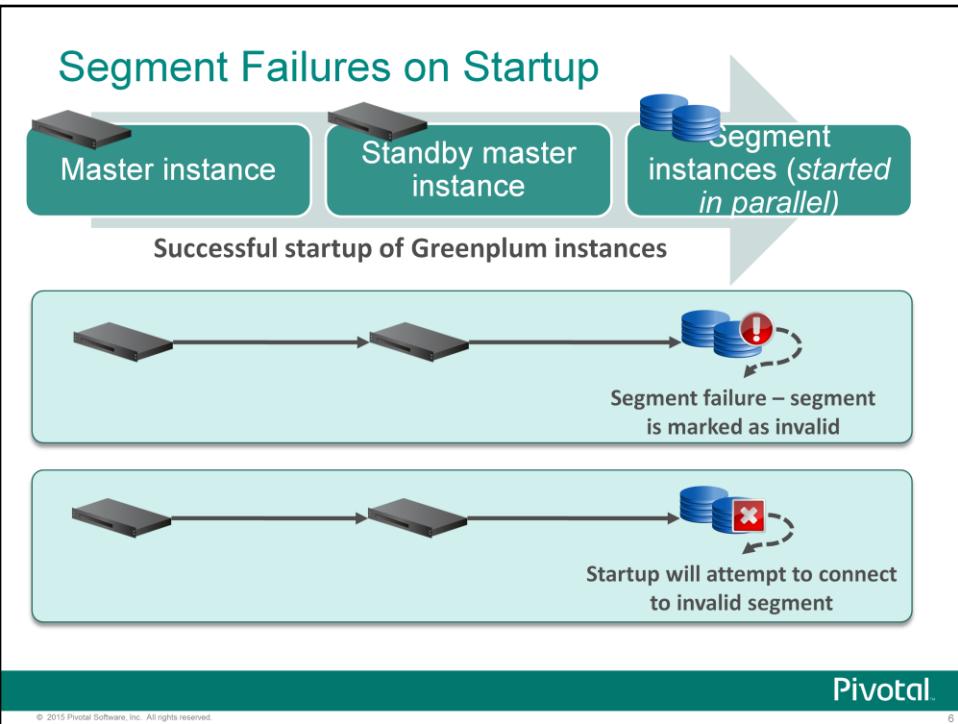
- **gpstart** – This command starts all instances in the array in the correct order.
- **gpstop** – This command stops all instances in the array in the correct ordering.

These scripts call the postmaster process on each segment instance with the correct flags. The master instance is always started last and is shut down first.

There may be cases where you want to start or stop only the master. This is called utility mode. In utility mode, you can:

- Connect to a database on the master instance only.
- Edit system settings in the system catalog, without affecting user data on the segment instances.

This is a very rare occurrence, but may be necessary in some situations, such as when reconfiguring the array to use different hosts.



If at startup a segment instance is not reachable:

- The system will mark it as invalid, meaning that the segment instance is offline and is not operational.
- The startup process skips segments marked as invalid and does try to start the server process, postmaster, on invalid segments.

This is a protective measure in case the segment instance needs some administrative action before it can safely be made operational again. When troubleshooting failed segment instances, you may at times need to force a startup of segment instances marked as invalid. For example, if a simple error, such as incorrect file permissions or an offline host, prevents startup of a segment instance, you will want to fix the problem that caused the error in the first place, and then retry the startup. Once the segment instance is up and running then it will be marked valid again next time the master tries to connect to it.

## Checking System State

The following commands are used to check the Greenplum state:

Action	Greenplum Application
Check system status	gpstate
Show complete system configuration and status	gpstate -s
Ports used by the system	gpstate -p
Segment mirror configuration	gpstate -m
Primary to mirror mapping	gpstate -c
Show details on primary/mirror segment pairs that have potential issues	gpstate -e
Obtain Greenplum Database version information	gpstate -i

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

7

The `gpstate` command displays status information about the Greenplum Database system, including:

- Current system configuration consisting of the number of configured segments, segment ports, hosts, data directories, primary/mirror status, mapping of the primary segment to its respective mirror segment.
- Segments that are marked as invalid.
- Mirror segment status.

## Recovering Down Segments

Recover Greenplum Database primary and mirror segments with the following:

Action	Greenplum Application
Recovers a primary or mirror segment instance that has been marked as down.	<code>gprecoverseg</code>
Rebalances primary and mirror segments by returning them to their preferred roles.	<code>gprecoverseg -r</code>
Performs a full copy of the active segment instance in order to recover the failed segment. The default is to only copy over the incremental changes that occurred while the segment was down.	<code>gprecoverseg -F</code>

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

8

In a system with mirrors enabled, the `gprecoverseg` utility reactivates a failed segment instance and identifies the changed database files that require resynchronization. Once `gprecoverseg` completes this process, the system goes into *resynchronizing* mode until the recovered segment is brought up to date. The system is online and fully operational during resynchronization.

A segment instance can fail for several reasons, such as a host failure, network failure, or disk failure. When a segment instance fails (primary or mirror), its status is marked as *down* in the Greenplum Database system catalog. In the case of a primary segment failure, its mirror is activated as the primary in *change tracking* mode. In order to bring the failed segment instance back into operation again, you must first correct the problem that made it fail in the first place, and then recover the segment instance in Greenplum Database using `gprecoverseg`.

## Checking for Data Distribution Skew

Check for data skew with the following:

- gp\_toolkit administrative schema offers two views:
  - gp\_toolkit(gp\_skew\_coefficients)
  - gp\_toolkit(gp\_skew\_idle\_fractions)
- To view the number of rows on each segment, run the following query:

```
SELECT gp_segment_id, count(*)  
FROM table_name GROUP BY gp_segment_id;
```

- Check for processing skew with the following query:

```
SELECT gp_segment_id, count(*) FROM table_name  
WHERE column='value' GROUP BY gp_segment_id;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

9

Performance can be inhibited if data is not equally distributed across all active segments. You can use any or a combination of the following to check for data skew:

- The gp\_toolkit schema provides two views that can be used to identify data skew.
  - The skccoeff column of the gp\_skew\_coefficients view is calculated as the standard deviation divided by the average. A higher value indicates greater data skew. The view also provides the table's name and namespace.
  - The siffraction column of the gp\_skew\_idle\_fractions view gives a percentage of the system that is idle during a table scan. This is an indicator of uneven data distribution or query processing skew. A value of .1 indicates a 10% skew. Tables with more than a 10% skew should be reevaluated.
- Data distribution can be viewed with the following SQL command:  

```
SELECT gp_segment_id, count(*) FROM table_name GROUP BY  
gp_segment_id;
```

This lists the number of rows on each segment. If all of the segments have roughly the same number of rows, the table is considered to be balanced.
- Processing skew, or the amount of time one query spends on acquiring data over another, can be determined with the following command:  

```
SELECT gp_segment_id, count(*) FROM table_name WHERE  
column='value' GROUP BY gp_segment_id;
```

This returns the number of rows returned by segment for the given WHERE clause.

## Log Files

Greenplum Database server log files:

- Are located in the data directory
- Are stored daily as CSV files
- Are stored on the master in  
\$MASTER\_DATA\_DIRECTORY/pg\_log
- Are stored on each segment in  
*/segment\_datadir/gpseg#/pg\_log*
- For management scripts, log files are located in  
*/superuser\_home/gpAdminLogs*
- Can be searched and filtered with the `gplogfilter` command

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

10

The log files for each Greenplum database server instance can be found in the data directory of that instance. If you are not sure of the location of the data directories, you can use `gpstate -s` to determine the location of the log files.

Log files for the Greenplum management scripts are written to the `gpAdminLogs` directory in the home directory of the superuser by default. These logs are on the master only. The log file for a particular script execution is appended to its daily log file each time that script is run.

Greenplum Database uses Postgres write ahead logging (WAL) for transaction logging on the master and each segment instance. WAL is on automatically. Refer to PostgreSQL documentation has more information about WAL.

## Logging Configuration Parameters

The following are commonly used for log configuration:

Log Parameter	Description
client_min_messages	Controls which message levels are sent to the client; accepts range from DEBUG5 to PANIC
log_min_messages	Controls which message levels are written to the server log; accepts range from DEBUG5 to PANIC
log_connections	Each successful connection is logged; accepts on off
log_statement	Controls which SQL statements are logged; accepts NONE, DDL, MOD, or ALL
log_rotation_age	Determines the maximum lifetime of an individual log file; accepts values in minutes
log_rotation_size	Determines the maximum size of an individual log file; accepts values in kilobytes
log_duration	Causes the duration of every completed statement to be logged; useful for query profiling

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

11

The slide lists several configuration parameters to set log levels, connection attempts, log rotation age and size, and query duration. There are numerous other parameters that can be used to fine tune logging to your requirements.

Refer to the *Greenplum Database Administrator Guide* for the complete list and details of each logging parameter.

## Greenplum Server Log Troubleshooting

Use the following tips to troubleshoot server problems:

- Check the master log to find the relevant log entry
  - Log lines have the format of:

```
timestamp | user | database | statement_id
| con# cmd# | :-MESSAGE _TYPE: <log_message>
```
  - The following is a sample of a log entry:

```
2006-08-19 19:00:58
PDT|lab1|names|11085|con107 cmd1|:-LOG:
statement: select * from topten where
year='2005' and gender='F' order by rank;
```
- Search the segment logs gpssh and gplogfilter

```
gpssh -f seg_host_file
=> source /usr/local/greenplum-db/greenplum_path.sh
=> gplogfilter -n 3 /gpdata/*/pg_log/gpdb*.log
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

12

To investigate an error or a log entry:

- First examine the master log file. This log file contains the most information.
- Each entry in the log has this information by default:
  - timestamp, user, database
  - *statement\_id* – Each statement issued to the Greenplum Database is assigned a *statement\_id*. All processes working on the same statement will have the same statement ID. The master increments this ID each time it receives a new SQL statement from the client.
  - *con#* – This number identifies a client session and connection.
  - *cmd#* – This is the command count number of the related session. Message type can be one of PANIC, FATAL, ERROR, WARNING, NOTICE, INFO, DEBUG1, DEBUG2, DEBUG3, DEBUG4, DEBUG5. It is then followed by the actual log message text.
- Use the *statement\_id*, connection number, and command count to find any related log entries on the segment instances. Note that a master log entry will not always have related segment entries.

The gplogfilter utility provides a way to search all segment logs at once.

## Maintaining the System Catalog and Reclaiming Disk Space

- Growth in the system catalog size:
- Can be caused by numerous database updates with `CREATE` and `DROP` commands
- Can be controlled with the `VACUUM` command or `vacuumdb` Greenplum application for regular system maintenance
- Can be controlled with `VACUUM FULL` for intensive system maintenance

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

13

Numerous database updates with `CREATE` and `DROP` commands cause growth in the size of the system catalog. This can affect system performance.

For example, after a large number of `DROP TABLE` statements:

- The overall performance of the system begins to degrade due to excessive data scanning during metadata operations on the catalog tables.
- The performance loss may occur between thousands to tens of thousands of `DROP TABLE` statements, depending on your system.

Greenplum recommends that you regularly run a system catalog maintenance procedure to reclaim the space occupied by deleted objects. If a regular procedure has not been run for a long time, you may need to run a more intensive procedure to clear the system catalog.

Periodically use `VACUUM` on the system catalog to clear the space occupied by deleted objects. This can be executed while the system is up, but should be run during a quiet period at off-peak hours.

A `VACUUM FULL` can be performed if regular system catalog cleaning is not performed regularly and the catalog has become bloated with dead space. This must be performed during system down time due to exclusive locks in use against the system catalog.

Periodic vacuuming reclaims unused space. `VACUUM FULL` is not recommended in large databases. It locks all tables and can take a really long time. A regular `VACUUM` is sufficient and is best to do at low-usage times. `ANALYZE` makes sure the planner always has the latest database statistics.

## Script for Reclaiming Disk Space

The following script can be used for periodic maintenance:

```
#!/bin/bash
DBNAME=<database_name>
VCOMMAND="VACUUM"
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
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

14

The script shown can be used to dynamically obtain the list of system tables found in the catalog for a specified database and execute a VACUUM against these tables. The script can be executed on a regular basis to remove bloat from the system catalog and remove the need to execute a VACUUM FULL on catalog tables. This is particularly useful for databases that are frequently updated, where tables are added and removed frequently.

## Checking Database Object Sizes and Disk Space

gp\_toolkit schema views for disk usage (in bytes):

Action	gp_toolkit View
Total size of all indexes for a table	gp_size_of_all_table_indexes
Size of a database	gp_size_of_database
Total size of an index	gp_size_of_index
Size of schemas in this database	gp_size_of_schema_disk
Disk size of a table	gp_size_of_table_disk
Uncompressed table size for append-only tables	gp_size_of_table_uncompressed
Amount of disk free, as determined by the OS df command	gp_disk_free



**Note:** Objects are shown by their object IDs. Look up the relation name in the pg\_class table.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

15

Monitoring disk space ensures that you have enough storage space on your segment hosts as your data grows. The gp\_size\_\* family of views available from the gp\_toolkit administrative schema, can be used to determine the disk space usage for a Greenplum database, table, schema, or index.

Objects are listed by their object ID and not by name. To check the size of a table or index by name, you must look up the relation name in the pg\_class table. For example:

```
SELECT relname as name, sotdsiz as size, sotdtoastsize  
as toast, sotdadditionalsize as other  
  
FROM gp_toolkit.gp_size_of_table_disk as sotd,  
pg_catalog.pg_class  
  
WHERE sotd.sotdoid=pg_class.oid ORDER BY relname;
```

## Detecting Bloated Tables and Tables with Missing Statistics

The following views provide information on bloated tables and tables missing statistics:

Action	gp_toolkit View
List tables that have bloat	gp_bloat_diag
List tables that do not have statistics	gp_stats_missing

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

16

Bloated tables or tables that do not have statistics can adversely affect query performance. It is important to use VACUUM or VACUUM ANALYZE to clean bloated tables and clean and generate statistics respectively. The following gp\_toolkit views provide information on the tables with regards to bloat and statistics:

- The **gp\_bloat\_diag** view lists the tables that have bloat. A table is bloated when the actual number of pages on disk exceeds the expected number of pages given the table statistics. Use VACUUM to reclaim disk space occupied by deleted or obsolete rows. This view is accessible to all users, however non-superusers will only be able to see the tables that they have permission to access.
- The **gp\_stats\_missing** lists tables that do not have statistics and therefore may require an ANALYZE or VACUUM ANALYZE be run on the table. Empty tables do not have statistics associated with them and will be displayed when selecting this view.

## Data Gathering for Troubleshooting with gpsupport



### System log information:

Process listing

Free memory

Hosts file

RPM packages

sysctl.conf

### Executed queries

Obtained from parsed log files

### Metadata

Schemas

Statistics

Configuration information

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

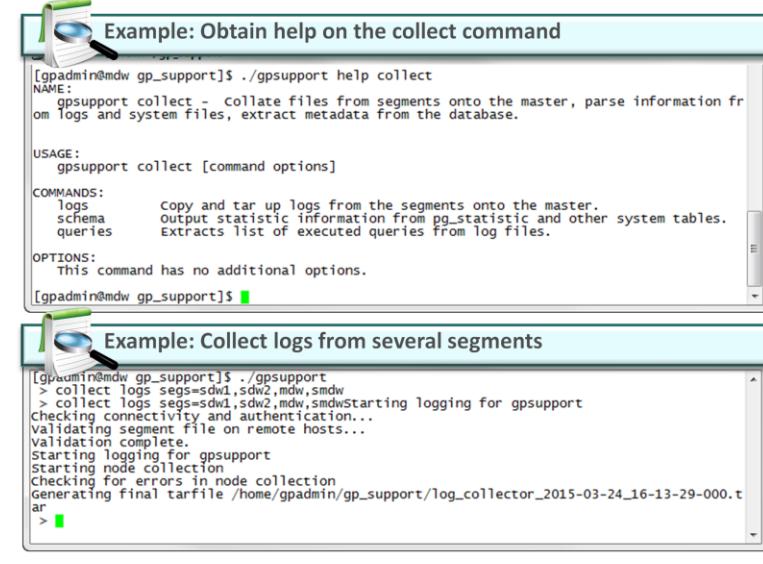
17

The `gpsupport` utility is a separate downloadable g-zipped compressed binary used to gather Greenplum Database information. The gathered data is used to troubleshoot and diagnose issues for GPDB 4.2.x and 4.3.x in a Linux operating system environment. The collected information can be provided to support personnel to help troubleshoot the environment.

Information is collected from the masters and segment servers and is stored to the master. The type of information or data that can be collected includes:

- Log files from segments, including, process listings, memory information, the hosts file, a listing of the packages installed on the system, and the contents of the `sysctl.conf` file. Information is collected from all segments you specify, master, standby, and segment servers, either directly on the command line or from a specified hosts file. The utility pushes an agent to the segment servers to collect information. This agent, `gpsupport_segment`, is pushed to the `/tmp` directory by default. Its location can be changed however.
- Executed queries collected from parsing log files. By default, the log files are collected from `/data/master/gpseg-1`.
- Metadata from the database, including schemas, statistics, and configuration information. This includes global objects, database statistics information collected from the `pg_statistic` catalog table, and planner statistics information collected from the `pg_class` catalog table.

## gpsupport Collection Examples



The screenshot shows two terminal windows demonstrating the use of the `gpsupport` command.

**Example: Obtain help on the collect command**

```
[gpadmin@mdw gp_support]$ ./gpsupport help collect
NAME:
  gpsupport collect - Collate files from segments onto the master, parse information from logs and system files, extract metadata from the database.

USAGE:
  gpsupport collect [command options]

COMMANDS:
  logs      Copy and tar up logs from the segments onto the master.
  schema    Output statistic information from pg_statistic and other system tables.
  queries   Extracts list of executed queries from log files.

OPTIONS:
  This command has no additional options.

[gpadmin@mdw gp_support]$
```

**Example: Collect logs from several segments**

```
[gpadmin@mdw gp_support]$ ./gpsupport
> collect logs segs=sdw1,sdw2,mdw,smdw
> collect logs segs=sdw1,sdw2,mdw,smdwstarting logging for gpsupport
  checking connectivity and authentication...
  validating segment file on remote hosts...
  Validation complete.
  Starting logging for gpsupport
  Starting node collection
  Checking for errors in node collection
  Generating final tarfile /home/gpadmin/gp_support/log_collector_2015-03-24_16-13-29-000.t
ar
>
```

Pivotal.

18

`gpsupport` provides both an interactive interface and a command line interface. Command usage is obtained with the `help` command. It can also be used to obtain additional information on other commands as shown in the first example where help is obtained on the `collect` command.

In the second example, `gpsupport` is executed in interactive mode. The `collect logs` command is executed with the `segs` option which allows you to specify which segment servers to collect information from. This is a comma separated list and overrides the `hostfile` global option which is used to specify the location of a host file. The host file requires that each host be listed on a line by itself.

The logs are saved to the directory from which you have executed the `gpsupport` command. You will require read and write permission to the directory you use for saving the captured information. You can specify a different output directory by using the `masterDataDir` option to this command. For example, to save the output from the second example to the `/tmp` directory, use the following command:

```
$ gpsupport collect logs segs=sdw1,sdw2,mdw,smdw
masterDataDir=/tmp
```

## Lab: Managing the Greenplum Database

In this lab, you use Greenplum utilities to check the state of the Greenplum environment and check for data and processing skew.

You will:

- Perform system administration tasks on the Greenplum environment

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

19

In this lab, you use Greenplum utilities to check the state of the Greenplum environment and check for data and processing skew.

# Module 6: Database Management and Archiving

## Lesson 1: Summary

During this lesson the following topics were covered:

- Starting and stopping the Greenplum Database
- Verifying the state of the Greenplum Database
- Using the Greenplum administrative schema to view which tables are exhibiting data skew
- Accessing log files and logging parameters
- Maintaining the system catalog and reclaim physical disk space
- Using `gpsupport` to gather troubleshooting data

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

20

This lesson covered several database management tasks, including starting, stopping, and verifying the state of the Greenplum Database. The administrative schema lets you find the tables that are exhibiting data skew, which can impact overall performance when querying tables. Other administrative tasks include accessing log files and logging parameters as well as maintaining the system catalog and reclaiming physical disk space, particularly in environments where there are a large number of deletes or updates for both system tables and user tables.

Troubleshooting involves gathering information from several different areas to analyze the underlying issue. The `gpsupport` utility is used to collate data from named segments from logs, metadata, system processes and information to create a complete picture of the system.

# Module 6: Database Management and Archiving

## Lesson 2: Backups and Restores

In this lesson, you examine how to backup and restore data from and to Greenplum Database.

Upon completion of this lesson, you should be able to:

- Describe the process of parallel backup
- Describe the process of parallel restore
- Describe the process of non-parallel backup
- Identify the commands used to perform backup and restoration of data
- Identify command options to perform incremental archival
- Identify the strategy to perform backup and restores with Data Domain

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

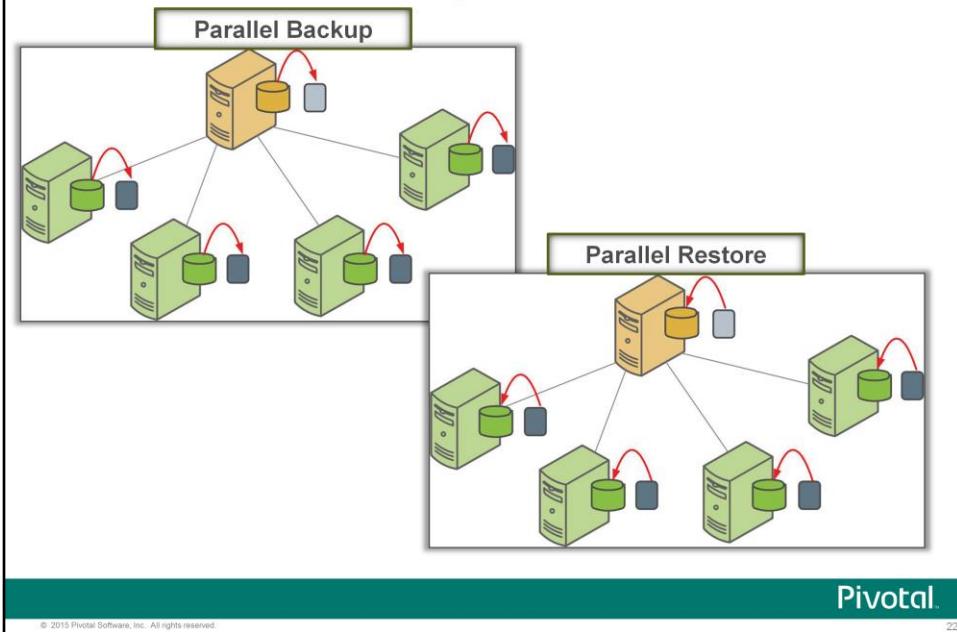
21

Regular backups improve the chances of recovering data in the event of a system failure or data corruption. Backups can also be used to migrate data from one system to another.

In this lesson, you will:

- Describe the process of a parallel backup.
- Describe the process of a parallel restore.
- Describe the process of a non-parallel backup.
- Identify the commands used to perform backup and restoration of data.
- Identify the options to support incremental backup and restore of append-only tables
- Identify the overall strategy for performing backup and restore with Data Domain.

## About Parallel Backups and Restores

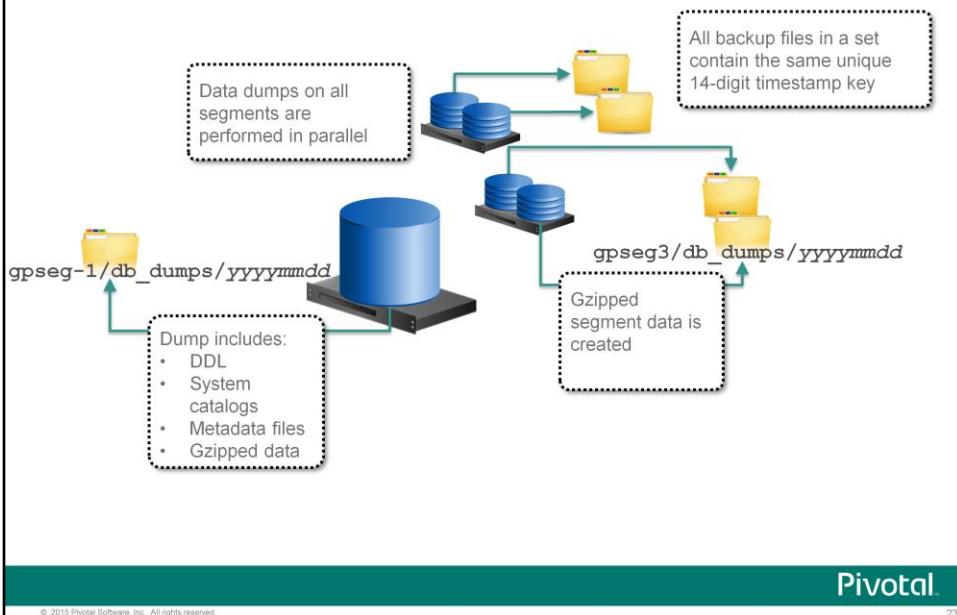


In Greenplum Database:

- You initiate a parallel dump operation from the master which dumps DDL for the database, schemas, and system catalog.
- This starts dump agents on the segments to dump all segment databases in parallel.

Likewise during a parallel restore, restore agents load each of the segment databases in parallel from their corresponding dump file as well as the data definitions for the master.

## Creating Parallel Backups



When performing a parallel backup, several actions occur:

- Every primary segment is dumped in parallel.
- A single dump file is created in the data directory for each of these segments. Within the data directory for the primary segment, two additional levels are created: `db_dumps/<yyyymmdd>`. The dump file created for the segment contains the data for an individual segment instance.
- The backup files are identified by a unique 14-digit timestamp key as part of the filename. All files within the same backupset are identified by this unique identifier.
- The master dumps the configuration of the database, or the system catalog, as well as DDL statements for recreating the database and schemas. This is saved to the master data directory under the same hierarchy.

As the dump files are created on the local file system for each segment, you must ensure there is sufficient disk space for the dump files both on the master and the primary segments.

## Dump Files Created During Parallel Backup

Master Segment Dump File	Description
gp_cdatabase_1_<dbid>_<timestamp>	CREATE DATABASE statement
gp_dump_1_<dbid>_<timestamp>	Database schemas
gp_dump_status_1_<dbid>_<timestamp>	Log file
gp_dump_1_<dbid>_<timestamp>_post_data	Post database setup (GUCs and search paths)
gp_dump_1_<dbid>_<timestamp>_ao_state_file	List of append-optimized tables
gp_dump_1_<dbid>_<timestamp>_co_state_file	List of column-oriented tables
gp_dump_<timestamp>.rpt	Database dump report
Primary Segments Dump File	Description
gp_dump_0_<dbid>_<timestamp>	Data for the segment
gp_dump_status_0_<dbid>_<timestamp>	Log file



**Note:** Each backup set shares the same unique timestamp.  
This timestamp is required for restoring a backup set.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

24

By default, the dump files are created under the data directory of the instance that was dumped. Files are saved to <data\_directory>/db\_dumps/yyyyymmdd.

On the master, dump files of the following are created:

- CREATE DATABASE statement
- DDL to recreate schema and database objects
- Log or status of the backup
- Post-database creation operations
- State files for append and column-oriented tables
- Report file that details the status of each segment with regards to the backups.

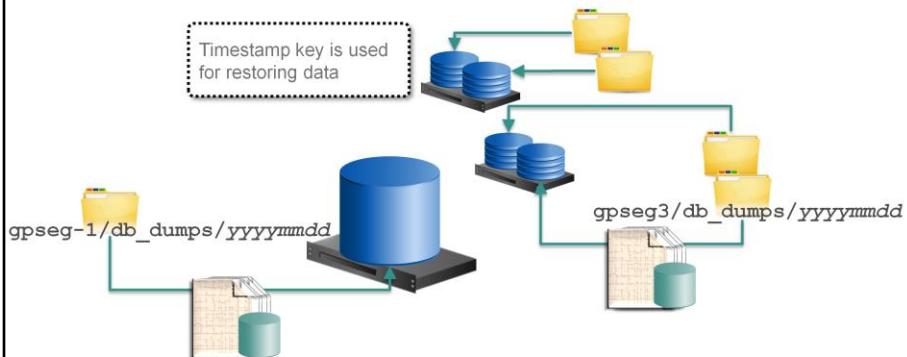
On the segments, dump files of the following are created:

- COPY statements and user data
- Log or status of the backup

You can identify which instance a dump file belongs to by the *dbid* in the dump file name. The master instance is always 1\_<dbid>, with the *dbid* usually being a 1 for the primary master. The segment instances are usually 0\_<dbid>. This is assigned in the gp\_configuration system catalog table.

Note that each dump file has the timestamp key which identifies the backup set. This timestamp is needed by a restore to identify the backup set.

## Performing Parallel Restores



Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

25

A parallel restore takes all the files associated with a backupset and restores the database objects and data to the distributed database in parallel. The unique timestamp key generated during the parallel backup is used to define the backupset. The data is validated and restored to the master and primary segments for the database captured in the backupset.

## Scheduling Routine Backups – gpcrondump

The `gpcrondump` utility:

- Can be called directly or can schedule using `cron`
- Should be scheduled on the master host
- Sends email notifications
- Flexible dump options
- Can copy configuration files
- Can dump system catalogs
- Can dump global objects
- Can include a post-dump script

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

26

Scheduling routine backups is a recommended practice to implement for disaster recovery sites. This includes automating nightly backups, moving backup files to a disaster recovery site, and providing a quick restoration option if needed.

The `gpcrondump` utility:

- Allows you to schedule routine backups of a Greenplum database using `cron`, a scheduling utility for Unix operating systems.
- Should be scheduled on the master host.
- Sends email notifications on the status.
- Provides a multitude of flexible dump options and checks.
- Copies the configuration files.
- Can dump only the system catalog.
- Can dump only the global objects.
- Can include a post-dump script for rebuilding the data.

## Restoring Archived Data

The `gpdbrestore` utility:

- Restores `gpcrondump` files
- Reconfigures for compression
- Validates the number of dump files
- Restores to active segment instances even with a failed segment
- Does not require you to retrieve the timestamp key
- Can restore from an archive host
- Does not require dump files to be placed on segments
- Identifies the database name automatically
- Detects the type of backup set available

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

27

The `gpdbrestore` utility provides the following additional functionality on top of `gp_restore`:

- Restores from backup files created by `gpcrondump`.
- Automatically reconfigures for compression.
- Validates the number of dump files are correct for primary only, mirror only, primary and mirror, or a subset consisting of some mirror and primary segment dump files.
- If a failed segment is detected and if `gp_fault_action=continue`, restores to active segment instances.
- Does not need to know the complete timestamp key of the backup set to restore. Additional options are provided to instead give a date, directory, or database name to restore.
- Lets you restore from a backup set located on a host outside of the Greenplum Database array (archive host). To do this, it ensures that the correct dump file goes to the correct segment instance.
- Identifies the database name automatically from the backup set.
- Allows you to drop the target database before a restore in a single operation.

## Incremental Backups

Incremental backups:

- Were Introduced in the 4.2.5 release and above
- Allow users to specify a point in time to restore the database to
- Are supported with:
  - Column- and row-oriented append-only tables
  - At the partition level of AO tables
- Back up an AO table if one of the following operations is performed:
  - ALTER TABLE, INSERT, TRUNCATE, DELETE, UPDATE
  - DROP and then re-create the table
- Cannot be used with Data Domain Boost

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

28

Starting in GP 4.2.5, options for incremental backup for append-optimized (AO) tables have been added to your support strategy. All other tables and database objects are backed up in a full backup mode.

Incremental backups are supported on both column-oriented and row-oriented AO tables. Support for incremental backups is also provided on AO tables that are partitioned.

When an incremental backup is executed on an AO partitioned table, only the partitions that have been modified will be captured. Changes to the table that are captured by incremental backups are tables that have been modified with one of the following:

- Alter table
- Insert
- Table truncate
- Dropping and recreating the AO table.
- Updating the table
- Deleting the table

If using Data Domain for the backup, the complete backupset must be on the Data Domain system. The complete backupset consists of the full backup along with the incremental backups. Note that Data Domain Boost is not supported with incremental backups.

## Managing Incremental Backups

- To create an incremental backup with `gpcrondump`, include the `--incremental` option
- You cannot use the Data Domain Boost options with a full backup if you plan to perform incremental backups.
- To restore data from an incremental backup you need a complete backup set, which consists of the following:
  - The last full backup before the current incremental backup
  - All incremental backups created between the time of the full backup the current incremental backup
  - The full backup and incremental backups need to be in the same directory location (the `gpcrondump -u` option will ensure this)

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

29

To run an incremental backup using the `gpcrondump` utility you must specify the option `--incremental`. As stated before incremental backups are not supported with DDBoost.

NOTE that if you are performing incremental backups, you have to ensure that your full backups are not executed with any of the DDBoost option found with the `gpcrondump` utility.

Restoring from an incremental backup requires the following

1. A Full backup has been performed (exists) before the current incremental that you want to restore from.
2. All incremental backups created between the time of the full backup the current incremental backup that you want to restore from exist.
3. All backups (full and incremental) are stored in the same directory location. A `-u` option can be used with `gpcrondump` to insure this.

## Incremental Backup Example

 Example: Creating a full backup of the faa database to /backupdir

```
$ gpcrondump -x faa -u /backupdir
```

 Example: Creating a series of incremental backups to /backupdir

```
$ gpcrondump -x faa -u /backupdir --incremental
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

30

The first command shown on the slide will perform a full backup of the faa database found in Greenplum. It will store all archive files in the directory backupdir.

The second command will perform an incremental backup on the faa database in Greenplum and place the archive files in the directory backupdir.

Note that the incremental backup only backs up changes performed on append-only tables found in the faa database. If no AO tables exist, or no AO tables have changed, a full backup will be performed.

## Incremental Backup Example (Cont)

- Full and incremental backups are saved in the user specified directory and named with an appropriate timestamps. After a series of backups you might see something like this in the backup directory:
  - 20120514054532 (full backup)
  - 20120714095512
  - 20120914081205
  - 20121114064330 (full backup)
  - **20130114051246**
- Restore a backup by specifying a point in time that corresponds to an existing incremental backup:

```
gpdbrestore -t 20130114051246 -u /backupdir
```
- The result of this command will be to restore the database using the last full backup (20121114064330) and the last incremental backup (20130114051246)

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

31

We can see that a number of full and incremental backup have occurred. Archive files are named using the following timestamp format (YYYYMMDDHHMMSS) . So you can see that the archive files ending in 4532 and 4330 are full backups, and the other backups are incrementals.

Running the gpdbrestore utility and specifying the archive file ending in 1246 will restore the database back to a time of the full backup ending in 4330 and the incremental backup ending in 1246.

NOTE: the other archive files in this example are not needed and will not be used with this restore command.

## Non-Parallel Backups and Restores

Non-parallel backups and restores:

- Are supported with the `pg_dump` and `pg_restore` utilities
- Are useful for migrating data to and from other DBMS

The `pg_dump` utility:

- Creates a single dump file
- Can be slow on very large databases
- Should be run at low-usage times
- Supports compression
- Can dump data as `INSERT` or `COPY` commands
- Includes the `DISTRIBUTED BY` statements in DDL with the `--gp-syntax` option

Pivotal

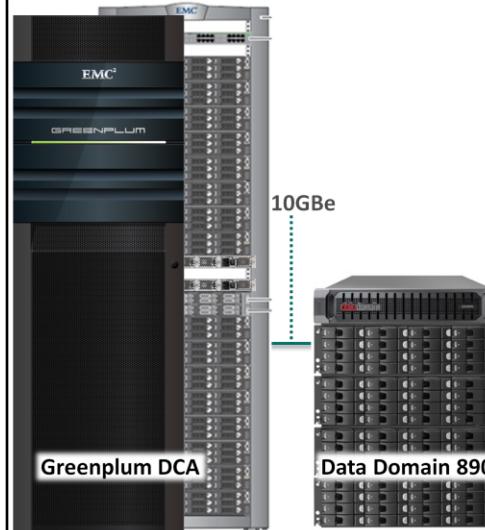
© 2015 Pivotal Software, Inc. All rights reserved.

32

The `pg_dump` utility creates a single dump file to the master server, instead of to the individual segments.

To use this command, you must have adequate file system space on the master to store the resulting backup file. This should be used when the system is quiet, as it has the potential to impact the overall performance, especially on very large databases.

## EMC Greenplum DCA and the Data Domain Solution



### Data Domain:

- Provides backup and recovery with Greenplum DB 4.1+
- Offers deduplication
- Supports:
  - NFS mounts with GPDB 4.1
  - DDBoost with GPDB 4.2
- Leverages `gpcrondump` and `gpdbrestore`
- Must be connected on the interconnect
- Provides access to each Greenplum Database instance

Starting with Greenplum Database 4.1 and 4.2, NFS mounted and native integration with Data Domain 880 and 890 was introduced to support backup and recovery services for the Greenplum Database. Data Domain offers a comprehensive backup management, offloading many of the duties a database administrator has with backup and recovery services to the device.

Deduplication offers a form of incremental backup not offered with the native Greenplum Database backup and recovery utilities.

There are two integration options offered for Greenplum Database with Data Domain:

- Backup and recovery with NFS mounted directories.
- Backup and recovery with DDBoost, a client library integrated with the database.

EMC Data Domain:

- Leverages the Greenplum backup utility, `gpcrondump`, to perform the backup and `gpdbrestore` for restores.
- Must be connected to the DCA on the interconnect switches. This enables 10 GbE connectivity between every DCA server and the Data Domain Appliance.
- Provides access to the Greenplum DCA using internal 10GbE network.

## Data Domain Integration: NFS Solution



### NFS integration:

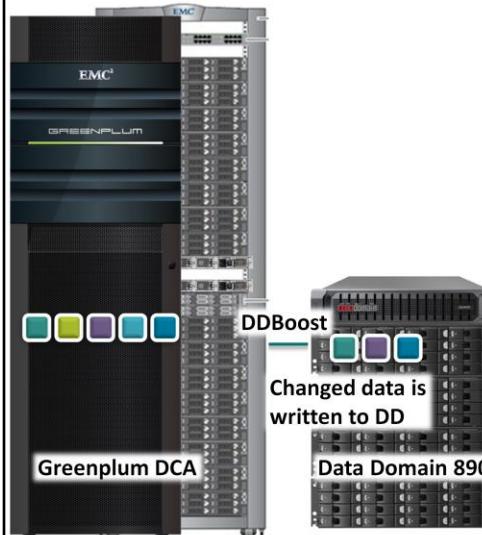
- Is available to GPDB 4.1 and 4.2
- Requires each server has its own mount point
- Performs deduplication and compression after data is sent over the network

Pivotal.

The NFS integration solution is available for both Greenplum Database 4.1 and 4.2. Once connected to the Data Domain, each segment server and the master server has its own mount point to the Data Domain appliance.

When performing a backup, data is sent over the network to the Data Domain appliance over the mount point. All deduplication and compression is applied after the information has been sent to the Data Domain appliance. While this is transparent to you, it can incur high costs for network bandwidth as all data is sent. At this point, there is no sense of incremental backup or deduplication until the data is fully sent to the Data Domain appliance.

## Data Domain Integration: DD Boost Solution



### DD Boost integration:

- Is a client library integrated with GPDB
- Uses native communication protocol
- Performs deduplication on the segments and master
- Only captures changed data
- Takes advantage of MPP design

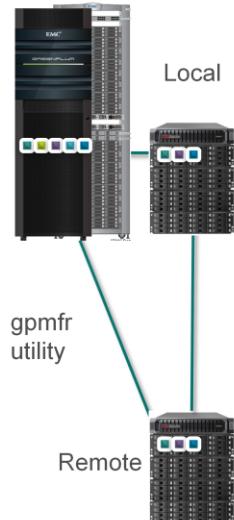
Pivotal

DD Boost is a client library integrated into the Greenplum Database, fully integrated with the Greenplum Database utilities for backup and recovery, and runs on all segments.

It is a native communication protocol which then performs deduplication on the segments and master themselves, reducing network traffic. The only data that is transferred is data that has changed. This immediately takes advantage of deduplication and provides a form of incremental backup. In addition to taking advantage of deduplication, the data is compressed before it is sent over the network.

DD Boost takes advantage of the Greenplum Database cluster by pushing deduplication processing to the segment hosts and utilizing the CPU cores available to perform deduplication and compression. The more CPU cores are available, the better the performance. This enables faster backup as the Greenplum Database scales.

## Data Domain Integration: Managed File Replication



- Managed File Replication (MFR)
  - Introduced with the 4.2.5 release of GP
  - Allows replication of Greenplum Database backup images stored on a local Data Domain system to a remote Data Domain system.
  - Data Domain login credentials have to be configured with `gpcrondump` utility on both the local and remote Data Domain systems.
  - The master segment must be able to connect to both the local Data Domain system and the remote Data Domain system.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

36

Managed File Replication is a feature found in Data Domain software. Collection Replication is the other type of replication supported.

Managed file replication directly transfers a backup image from one Data Domain system to another, one at a time on request from the backup software.

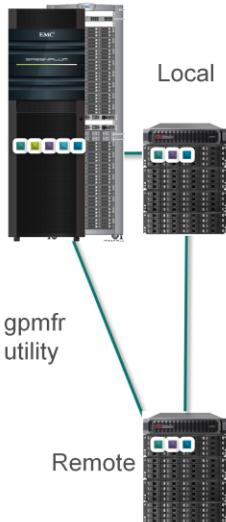
Collection replication performs whole system mirroring in a one-to-one topology continuously transferring changes in the underlying collection, including logical directories and files of the Data Domain filesystem.

Prior to Greenplum 4.2.5 only collection replication was supported. In the 4.2.5 and above releases of Greenplum you know can perform a Managed File Replication (MFR).

MFR allows you to take backup data sets (a set of archive files) and copy them over to a remote Data Domain system (a disaster recover site). The remote site data sets can then be used to perform restore operations of the Greenplum Database.

The Master server must be able to communicate with both Data Domain systems (local and remote) and do require additional setup steps. These additional steps require running the `gpcrondump` utility with certain `--ddboost` options to establish user credentials on the Data Domain systems.

## Data Domain Integration: Managed File Replication



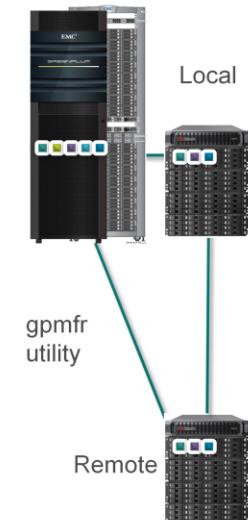
- The `gpmfr` utility manages the Greenplum Database backup sets on the local and remote Data Domain systems.
- The `gpmfr` utility provides these capabilities:
  - Lists the backup data sets that are on the local or the remote Data Domain system.
  - Replicates a backup data set that is on the local Data Domain system to the remote system.
  - Recovers a backup data set that is on the remote Data Domain system to the local system.
  - Deletes a backup data set that is on the local or the remote Data Domain system.

Pivotal

A new utility called `gpmfr` has been created that lets you manage your backup data sets on the local and remote Data Domain systems. `gpmfr` allows you to list, replicate, recover, and delete your backup sets on both Data Domain systems.

37

## Data Domain Integration: MFR Example



The following example replicates the latest backup set on the local Data Domain sever to the remote server. The maximum number of I/O streams that can be used for the replication is 30.

```
gpmfr --replicate  
LATEST --max-streams  
30
```

Pivotal

The following example replicates the latest backup set on the local Data Domain sever to the remote server. The maximum number of I/O streams that can be used for the replication is 30.

© 2015 Pivotal Software, Inc. All rights reserved.

38

## Comparing the EMC Greenplum DCA Integration Solutions

Feature	NFS	DD Boost
Deduplication	Deduplication on Data Domain appliance	Deduplication on Greenplum DB segment and master instance
CPU Usage on Segments	As needed for NFS	Increased CPU usage on GPDB due to de-duplication and compression
Network Utilization	All data is sent over the network	Only changed, deduplication data is sent over the network
Scalability	Increasing the number of racks can result in saturation of DD appliance or network	Minimal data transfer
Management	Each segment server and master server requires its own mount point	Integrated native solution with no static system configuration
Backup Performance	Full backup	Initial backup is full; follow-on backups are incremental
Data Domain Replication	Directory level	Collection and Managed File

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

39

To use the Data Domain for backup and restore:

- All servers in the DCA must have NFS access to the Data Domain. Data Domain provides a dedicated NFS export to each of the servers in the DCA. For example, Data Domain will export the directory, /backup/DCA-01/mdw, which is available to the master server as /backup. Each segment will have its own directory available as well.
- The superuser account, gpadmin, must have full read and write access to the backup directory.

Once the NFS configuration has been completed, the Greenplum backup and restore utilities can be used to write to or read from the mount points.

## Lab: Backups and Restores

In this lab, you use the Greenplum backup utilities to process a backup in your environment.

You will:

- Create and retrieve full backups
- Create and retrieve incremental backups

Pivotal

In this lab, you use the Greenplum backup utilities to backup databases in your environment. You will perform both incremental and full backups of your data.

# Module 6: Database Management and Archiving

## Lesson 2: Summary

During this lesson the following topics were covered:

- Process of parallel backup
- Process of parallel restore
- Process of non-parallel backup
- Commands used to perform backup and restoration of data
- Command options to perform incremental archival
- Strategy to perform backup and restores with Data Domain

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

41

This lesson covered various archival topics, including the difference between parallel and non-parallel backup and restore and how to perform each of these using the Greenplum backup and restore utilities. Greenplum backup utilities include options both for incremental support for append-only storage as well as storing and restoring data to a Data Domain.

## Module 6: Summary

Key points covered in this module:

- Performed system administrative tasks, including managing and checking the state of the Greenplum Database and its data as well as checking the distribution of data
- Performed backup and restoration of Greenplum data from and into the Greenplum Database respectively

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

42

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

- Perform system administrative tasks, including managing and checking the state of the Greenplum Database and its data as well as checking the distribution of data.
- Perform backup and restoration of Greenplum data from and into the Greenplum Database respectively.

# Module 7: Data Modeling and Design

This module more closely examines how you model and design your data for the Greenplum Database. Upon completion of this module, you should be able to:

- Identify and describe the data models used in data warehousing and describe how data is stored in Greenplum
- Distribute and store data in Greenplum using a distribution key, partitioning, and constraints

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

1

Key decisions must be made on how data is designed for the Greenplum Database. These decisions can affect business decisions that are based on data and determine the overall performance for accessing data.

In this module, you will:

- Identify and describe the data models used in data warehousing and describe how data is stored in Greenplum
- Distribute and store data in Greenplum using a distribution key, partitioning, and constraints

# Module 7: Data Modeling and Design

## Lesson 1: Data Modeling

In this lesson, you more closely examine the models used in the Greenplum Database.

Upon completion of this lesson, you should be able to:

- Define the three data models, including the logical data model, enhanced logical data model, and the physical data model
- Identify common data models used in data warehouses
- List three ways data are segregated

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

2

Data modeling is a key area of data warehouse design. Three models were presented earlier, with benefits for OLAP and OLTP. It is important to gather requirements and build the data model before implementing it. A poorly designed database can have repercussions beyond performance issues, affecting the expected outcome of queries, and therefore, affecting business decisions.

Upon completion of this lesson, you should be able to:

- Define the concepts, logical, enhanced logical, and physical data models and implement appropriate data models for Greenplum.
- Identify the common data models used in data warehouses.
- List three ways in which data are segregated.

# Data Models

## Logical data model

- Refines business requirements
- Defines the entities, attributes, and relationships
- Uses business names
- Is independent of technology
- Can be thought of as an external model (end user's view)
- May be normalized

## Enhanced logical data model

- Refines objects defined in the logical data model (LDM)
- Represents a global view of the database as viewed by all
- Most widely represented by the ER model

## Physical data model

- Operates at the lowest level of abstraction
- Implements the logical model
- Describes how data is saved
- Is dependent on the DBMS
- Defines database objects
- May be denormalized

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

3

## Data modeling:

- Is the first step in designing the database and refers to the process of creating a specific data model for a determined problem.
- Is an iterative process where you continuously refine the requirements and the details of the data model.
- Acts as a communication tool to show, in great detail, how business requirements translate into business decisions, and the business rules each organization creates and uses can affect another organization or the business overall.
- Shows the different ways in which data is viewed by individuals and organizations.
- Provides the basic building blocks for entities, attributes, relationships, and constraints .
- Realizes a physical model where data is represented at the lowest level of abstraction.

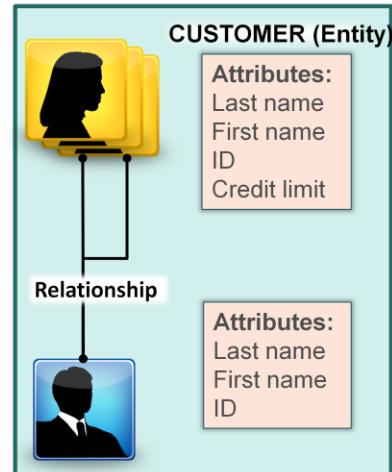
Data design flows from creating and refining the logical business model, where you define the basic building blocks based on the requirements gathered from stakeholders, to an enhanced logical model, where you further refine the logical data model with detailed information on the objects, to the physical design, where you implement the logical data model.

**Note:** Normalization is concerned with relationships. It is a process for evaluating and correcting table structures or models to minimize data redundancies. Denormalized data is not concerned with redundancies.

## Data Modeling Terms

Data modeling terms include:

- Entity:
  - Can exist on its own
  - Can be uniquely identified
- Attribute – Defines a property of the entity
- Relationship – How entities relate to each other
- Constraint – A restriction placed on the data
- Primary Key – A single or combination of attributes that uniquely identifies the entity



Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

4

The following terms are commonly used in data modeling and describe the basic building blocks of the model:

- **Entity** – This describes the unique person, place, thing, or event about which the data are collected and stored. It can exist on its own.
- **Attribute** – The attribute is a characteristic or description of the entity. It defines the property of that entity.
- **Relationship** – The relationship shows how the association between entities and how they relate to each other.
- **Constraint** – The constraint is a restriction placed on the data, helping to ensure integrity. For example, the credit limit for the customer must be entered in numbers greater than or equal to 10,000.
- **Primary key** – This term describes the attribute or combination of attributes that uniquely identifies the entity. For example, the combination of first and last name may be enough to describe the customer. However, it is more likely that the ID attribute is sufficient and unique enough to identify the customer, as long as there is a constraint on that attribute requiring that no two IDs are the same.

# Logical Data Models



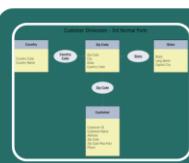
## Star Schema

- Easiest for users to understand
  - Single layer of dimensions
  - Divides data into fact and dimensions
  - May have data redundancy



## Snowflake Schema

- Greater reporting ability
  - Can break up large dimensions into manageable chunks
  - May have redundant data



### → Third Normal Form (3NF)

- Most flexible option
  - No redundant data
  - Most difficult for users to use

Pivotal™

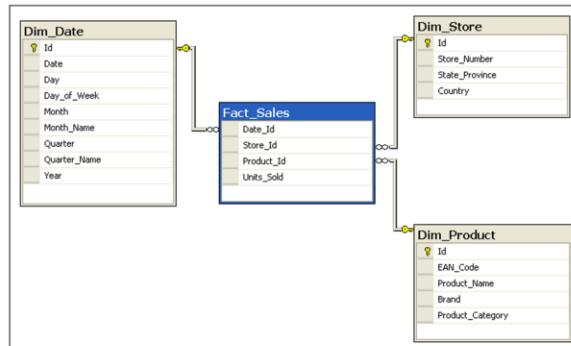
© 2015 Biuntal Software, Inc. All rights reserved.

65

One of the more commonly used models in a data warehouse and OLTP systems include:

- **A star schema** – This is one of the easiest dimensional models to understand as it represents data as a single layer of dimensions, dividing data into facts and dimensions.  
There can be one or more fact tables represented in this model. The fact table contains measures of interest, while the dimension or lookup tables, which connect to the fact table, provide detailed information on the attributes. Lookup tables do not have a relationship with each other, instead, relying on their relationship to the fact table.
  - **A snowflake schema** – The snowflake schema dimensional model is an extension of the star schema and offers greater reporting capabilities. It divides the large fact table into more manageable portions, by normalizing the dimensional table into multiple lookup tables. Each lookup table may represent another level in the dimensional hierarchy. This type of model may incur redundant data.
  - The third normal form (3NF) – The 3NF form.

## Star Schema Dimension Model



A dimension:

- Has a known entry point to a fact
- Is a defined attribute of a fact
- Has a simple Primary Key

A fact:

- Is measurable
- Relates to a specific instance
- May have compound primary Key

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

6

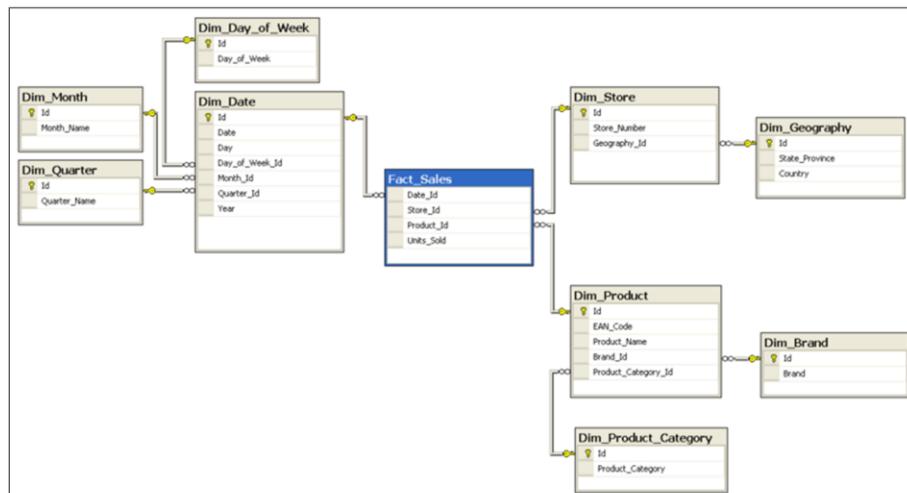
The dimension table:

- Also known as a lookup table, has a known entry point to a fact.
- Provides a description of the fact. For example, the `Dim_Store` dimension table contains descriptions of the store, including the store number, ID, and location.
- Has a single primary key for each record. This primary key uniquely identifies the store in the `Dim_Store` table.

The fact table:

- Combines the information provided in the lookup tables to provide a measurable account of a unique record.
- The fact table in this example, `Fact_Sales`, identifies the date, store ID, product ID, and number of units sold for each unique record.
- A unique record can be determined by a combination of primary keys, or a compound primary key. It may not be enough to uniquely identify a transaction or sale just by the `Date_Id`. It may require that all four fields in the table uniquely identify a sale.

## Snowflake Schema Dimensional Model



Pivotal.

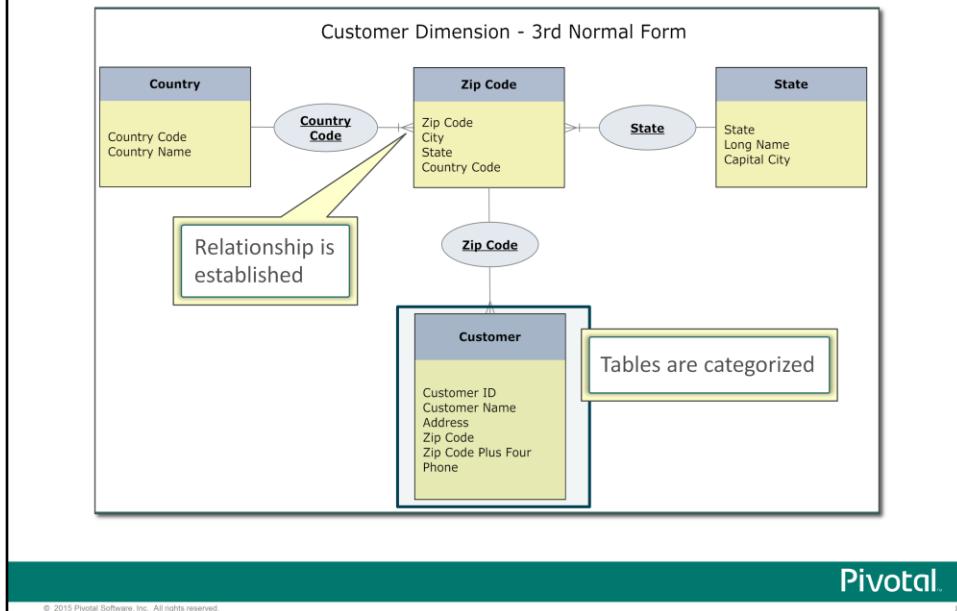
© 2015 Pivotal Software, Inc. All rights reserved.

7

In the snowflake schema, the dimensional tables are further normalized into multiple lookup tables. Your data is no longer represented in a single dimension, but is now shown as a dimensional hierarchy.

Snowflake schema offer some improvement in query performance due to minimized disk storage requirements and joining on smaller tables. This comes at a price however. Additional maintenance is required due to the increased number of lookup tables generated by this type of model.

## Third Normal Form (3NF) Model

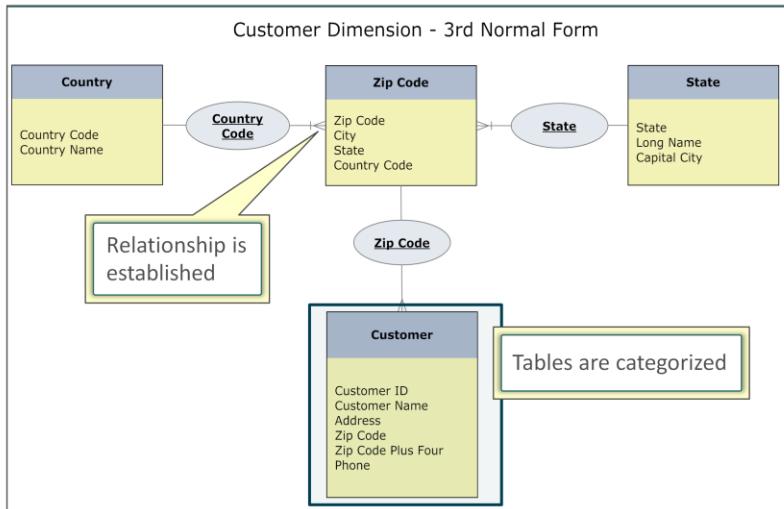


The third normal form (3NF) design normalizes data to remove redundancy and groups data together by subject areas or categories. This can be difficult for users to understand, but it is more suited to capturing complex the complexities of real world entities and the resulting data. This type of model can be used to help identify patterns and trends instead of simply providing reporting support and support for well established relationships.

Getting this form can sometimes be an iterative process, but can be accomplished by going through lower forms:

- 1NF requires the existence of a primary key and the identification of dependencies. The primary key can be a combination of multiple fields. This primary key must be able to uniquely identify each record. Dependencies are fields that are dependent, in some way, on the primary key.
- 2NF requires the table first be in 1NF and there are no partial dependencies. A partial dependency is when an attribute is dependent on a portion of the primary key. If a primary key consists of three fields, a partial dependency occurs if another field is dependent on only one of those three fields. A table with a primary key that contains only a single attribute is automatically in 2NF.
- 3NF requires that the table be in 2NF and contains no transitive dependencies. A transitive dependency exists when an attribute, other than one involved in the primary key, is dependent on another attribute that is not a part of the primary key.

## Third Normal Form (3NF) Model (Cont)



Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

9

The example on the slide shows the following relationships:

- A country can have multiple zip codes.
- A state can have multiple zip codes.
- In a zip code, you can contain multiple customers.

## Data Warehouse and Logical Data Models

### Dimensional Approach in DW

- Star and snowflake schemas are the most common in DW implementations
- Offers fast retrieval of large numbers of rows
- Easier to aggregate
- Intuitive to end users
- Difficult to maintain data integrity
- Can be difficult to modify

### Normalized Approach in DW

- Simple for data loading
- Easier to modify
- Easier to maintain data integrity
- Difficult to understand and use
- Can complicate query writing
- Slower performance due to multiple table joins on larger tables

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

10

There are benefits to using one type of model over another. Dimensional models can offer better query performance while normalized models can provide insight into complex data. Real-time data may be easier to implement in 3NF than in dimensional models, but dimensional models offer greater reporting capabilities.

## Enhanced Logical Data Model

An enhanced logical data model:

- Provides more detail than the Logical Model
- Defines the cardinality of the data attributes
- Defines the anticipated initial data volume (in rows)
- Defines the anticipated data growth over time

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

11

Often represented as entity relationship (ER) diagrams, the enhanced logical data model extends the logical data model by providing more details about the entities defined earlier, including:

- Defining the cardinality of data attributes, which is the relationship of a given table to another. Cardinality can be defined as a one-to-one relationship, one-to-many relationship, or many-to-many relationship.
- Defining the anticipated initial volume of data expected to be loaded into systems.
- Defining the anticipated growth of data over a period of time.

## Enhanced Logical Data Model Example

The following is an example of the enhanced logical data model:

Entity: Store  
Source: SOLS (Stores On-Line System)  
stores table  
Row Count at Source: 35  
Anticipated Annual Growth Rate: 100

Attribute	Source	Uniqueness	Data Type	Data Example
Store ID	Stores	Unique	Small Integer	1,2 ... N
Store Name	Stores	Non	Character	Tom's Groceries
...				
Has Pharmacy	Stores	Non	Character	T,F

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

12

In the example shown, the detailed information is provided about the entity, **Store**, including the source for this data, the anticipated row count and the anticipated growth rate for that object.

The entity is described by its attributes in the table at the bottom of the slide. Data types and examples of the expected values are included in the model.

## Physical Data Model

The physical data model is represented with the following:

- Table – The physical manifestation of an entity
- Columns – How the attributes of the entity are physically represented
- Constraints
  - Foreign key
  - Uniqueness
  - Primary key
  - Null / Not Null
  - Check of values

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

13

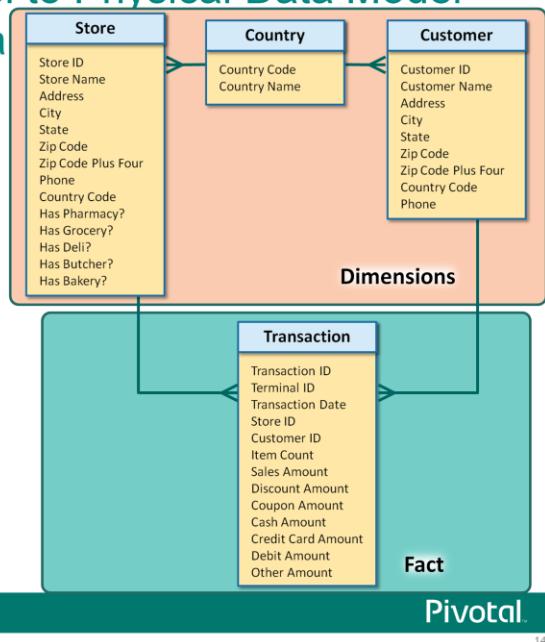
The physical data model defines the objects that will be created in the database. This includes:

- **Table** – A table is a physical representation of the entity defined in the logical data model.
- **Columns** – Each of the attributes defined for the entity are displayed as columns in the table.
- **Constraints** – Constraints are any restrictions on the attributes as they are entered in the table. Constraints can be any of the following:
  - Foreign keys which shows the relationship from one table to another.
  - Uniqueness where you specify that a value must be unique.
  - Primary key where you define the set of attributes that uniquely identify each record.
  - Null or not null where the column may or may not have a null value.
  - Check of values where you define acceptable values for the column.

## Logical Data Model to Physical Data Model – The Logical Data Model

In this example:

- Dimensions defined are:
  - Store
  - Country
  - Customer
- Fact is defined as Transaction



This example shows how a logical data model in 3NF is implemented in a physical data model.

On this slide, three dimension tables are defined:

- Store
- Country
- Customer

The fact table is defined as Transaction.

Relationships between the entities are as follows:

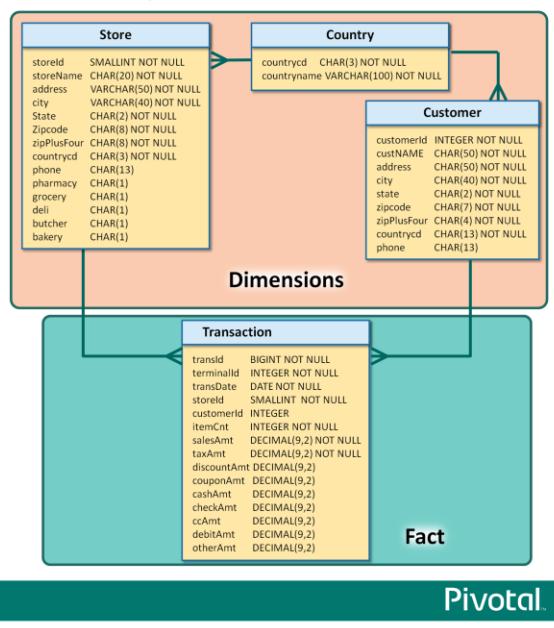
- Country has a one-to-many relationship with Store
- Country has a one-to-many relationship with Customer
- Customer has a one-to-many relationship with Transactions
- Store has a one-to-many relationship with Transactions

# Logical Data Model to Physical Data Model

## – The Physical Data Model

In this example:

- Entities become tables
- Attributes become columns
- Relationships may become foreign key constraints



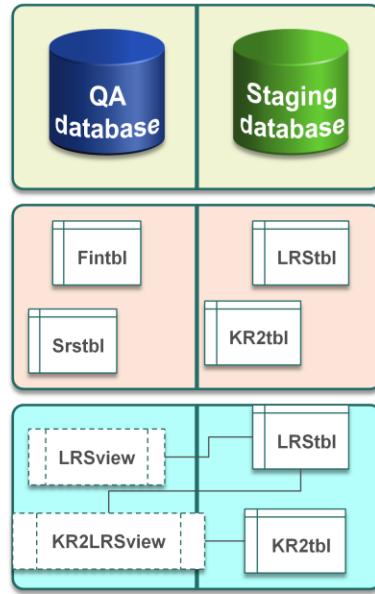
In the physical data model for the logical data model shown on the previous slide:

- Any of the entities defined as either a dimension or fact is now defined as a table.
- Attributes defined for each of those entities become columns.
- Relationships defined between the entities can become foreign keys.

## Data Segregation

Data can be segregated by:

- Databases:
  - You can define multiple in a Greenplum system
  - They do not share data in Greenplum
- Schema – A physical grouping of database objects
- Views:
  - All or some table columns may be seen
  - Allows for user defined columns which is instantiated at run time



Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

16

Data segregation can occur at any of the following levels:

- **Databases** – You can define multiple databases in a Greenplum system. While databases can share roles, they can not view or access data in another database.
- **Schema** – Schemas are used to physically group database objects with each other. A role with appropriate privileges can access data in different schemas. However, the data is physically separated and can be accessed by putting the schema name in front of the object name.
- **Views** – Views provide a functional view of data, hiding the constructs of the table from the user. A view can be created that allows three of five columns of data to be visible to a user.

## Lab: Data Modeling

In this lab, you examine data models and implement the design in the database.

You will:

- Create the datamart database and database objects

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

17

In this lab, you examine data models and implement the design in the database.

## Module 7: Data Modeling and Design

### Lesson 1: Summary

During this lesson the following topics were covered:

- Defining the three data models, including the logical data model, enhanced logical data model, and the physical data model
- Common data models used in data warehouses
- Methods in which data are segregated

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

18

This lesson provided an introduction to various data models, including logical, enhanced, and physical data model. Going through the process of defining your data based on the data model allows you to map business requirements to database models. The lesson reviewed the data models commonly found in data warehouses and the methods used to segregate data.

## Module 7: Data Modeling and Design

### Lesson 2: Physical Design Decisions

In this lesson, you examine the physical design decisions you will make while creating the database and its objects.

Upon completion of this lesson, you should be able to:

- Select the best distribution key used for distributing data
- Check for data skew
- Identify the benefits of partitioning a table and when to partition a table
- Determine partitioning candidates
- Select appropriate data types for your data
- Define constraints on tables and columns

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

19

You must take several key items into account when creating the physical design for your database. These decisions have a great impact on the overall performance of data storage and queries.

Upon completion of this lesson, you should be able to:

- Select the best distribution key that will be used for distributing data.
- Check for data skew.
- Identify the benefits of partitioning a table and when to partition the table.
- Determine partitioning candidates.
- Select appropriate data types for your data.
- Define constraints on tables and columns.

## Key Design Considerations

The following are key design considerations to account for:

 Data distribution and distribution key selection

 Checking for data skew

 To partition or not

 Choosing appropriate data types

 Defining constraints

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

20

Greenplum provides several options meant to improve overall query performance for your database implementation. This requires that you make informed decisions when defining the physical data model for your database. You must take into account the following to help you successfully design your system:

- **Data distribution and distribution key selection** – Select the best distribution key that matches your queries. Try not to select columns that are rarely used or are not in the distribution policy for the tables. This will result in slow queries.
- **Data skew** – If you have cases where queries are performing slowly, check for data skew. Data may be unbalanced across the segments. Verify that it is more evenly distributed. If not, consider your options for rebalancing the data.
- **To partition or not** – The decision to partition tables is based on your data and the speed of queries. It takes large tables and breaks them into more manageable pieces which can provide an improvement in performance.
- **Choosing appropriate data types** – Data types can affect storage, which can affect database size, and ultimately, affect performance. Choose the smallest data type that can accommodate your data.
- **Defining constraints** – Use constraints to set parameters around your data and give you more control over data in your tables.

## Primary Key and the Distribution Key

Consider the following:

- A primary key is a logical model concept which allows each row to be uniquely identified.
- A distributed by key is a physical Greenplum database concept which determines how a row is physically stored.
- The distributed by key can differ from the primary key.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

21

Primary keys in the logical model concept is usually an issue for Terradata users. Understanding that a primary key is a logical model concept where a distributed key is a physical storage concept is key to understanding how your data works.

A well designed Greenplum schema could have a large percentage of tables where the physical distributed by key is different from the logical primary key (tables with common distributed by keys provide faster join operations by reducing number of motion files).

## Compare and Contrast – Distribution and Primary Keys

Primary Key	Distribution By Key
Logical concept of data modeling	Physical mechanism for storage
Optional for Greenplum (becomes the distribution index if specified)	Requirement for every Greenplum table
No limit to number of columns	Defined at table creation time or modifiable later
Documented in data model	Can be non-unique
Must be unique	Can uniquely or non-uniquely identify each row
Uniquely identifies each row	Can be NULL
Value can not be changed	Defines a storage path
Can not be NULL	Chosen for distribution and performance
Does not define an access path (DBMS independent)	
Chosen for logical correctness	
Combination of UNIQUE and NOT NULL constraint	

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

22

This slide compares the differences between a primary key versus a distribution key. While distribution keys can uniquely identify rows, most times, they do not.

Primary keys are optional in Greenplum. If you do not specify a distribution key, and a primary key exists, it will, by default, be used as the distribution key. It can ensure the most even distribution of data.

## Data Distribution

In defining distribution methods, there are two options:

- Column distribution with the `DISTRIBUTED BY` clause
- Random distribution with the `DISTRIBUTED RANDOMLY` clause

```
CREATE TABLE tablename (
    column_name1      data_type NOT NULL,
    column_name2      data_type NOT NULL,
    column_name3      data_type NOT NULL DEFAULT default_value,
    . . .
    [DISTRIBUTED BY (column_name)]           hash algorithm
    [DISTRIBUTED RANDOMLY]                  random algorithm
```



**Note:** Every table in the Greenplum database has a distribution method, whether you select it or not.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

23

Every table in a Greenplum database has a data distribution method. The `DISTRIBUTED` clause specifies the distribution method for a table. There are two distribution methods:

- Column distribution is designated by specifying a column name in the `DISTRIBUTED BY` clause. The `DISTRIBUTED BY (column name)` clause will distribute the data across all segment instances using a hash algorithm.
- Random distribution which is designated by specifying the `DISTRIBUTED RANDOMLY` clause. The `DISTRIBUTED RANDOMLY` clause will distribute the data across all segment instances using a random algorithm.

## Distribution Key Considerations

Consider the following:

- A distribution key can be modified, even after the table has already been created.
- If a table has a unique constraint, it must be declared as the distribution key.
- User defined data types and geometric data types are not eligible as distribution keys.



**Note:** You should choose a distribution key that is unique for each record.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

24

While the distribution policy on a column can be changed, care should be taken to do so. This action redistributes data recursively through child tables.

A table can have one unique constraint and the unique column (or set of columns). If a table has a unique constraint it must be specified as the distribution key for the table.

Columns that contain a user defined data type or geometric data type are not eligible to be used as the distribution key for a table.

If your primary key consists of 13 columns, your distribution key should include those 13 columns at the very least.

## Default Distribution Use

Consider the following when creating tables:

- If a table has a primary key and a distribution key is not specified, by default the primary key will be used as the distribution key.
- If the table does not have a primary key and a distribution key is not specified, then the first eligible column of the table will be used as the distribution key.
- User defined data types and geometric data types are not eligible as distribution keys.
- If a table does not have an eligible column then a random distribution is used.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

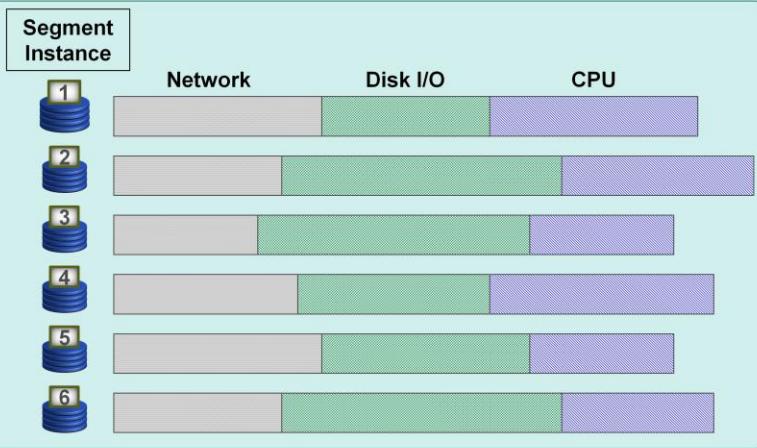
25

Every table in a Greenplum database has a data distribution method. You should consider not using the system default distribution as it may not result in an even distribution method. You should be fully aware of the data that will be loaded or inserted so that you make an informed decision on the distribution policy.

When creating tables, the following actions should be accounted for:

- If a distribution method is not specified and the table has a primary key defined then the primary key will be used as the distribution key. If the primary key is chosen well, this can ensure even distribution. However, you should determine for yourself whether the primary key makes a good distribution key.
- If a distribution method is not specified and the table does not have a primary key defined then the first eligible column will be used as the distribution key. Allowing the system to choose the first column as the distribution key can lead to uneven distribution of data. Once again, you must be responsible for selecting a distribution key based on known data and queries.
- User defined data types and geometric data types are not eligible distribution keys.
- If the table does not have a column of an eligible data type, the rows by default are distributed using a random distribution. While this will more evenly distribute your data, it may not be the right selection for your type of data and queries.

## Distributions and Performance



**Note:** To optimize performance, use a hash distribution method (DISTRIBUTED BY) that distributes data evenly across all segment instances.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

26

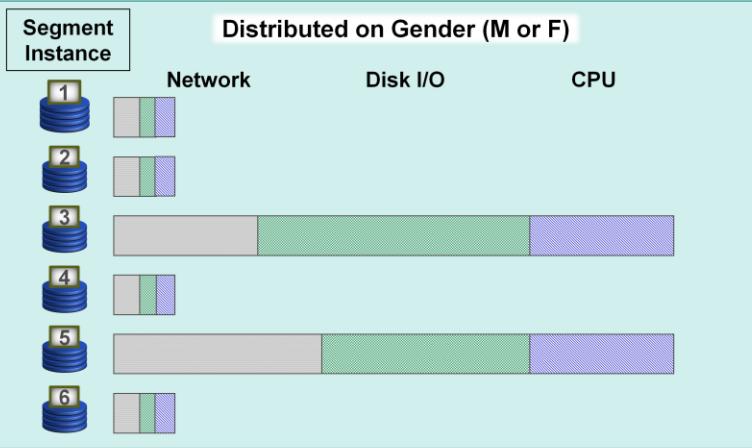
For the best possible performance, use a distribution method that distributes data evenly across all segment instances. In an MPP environment, overall response time for a query is measured by the completion time for all segment instances. If the data is skewed, then the segment instances with more data will have a longer completion time. The optimal goal is for each segment instance to have a comparable number of rows and to perform approximately the same amount of processing. Remember, your database is only as fast as its slowest segment.

For large tables a hash distribution that evenly distributes table rows across all segment instances is desired. While a random distribution will create a more even distribution across all segment instances, the significant performance gains of co-located joins is lost. Co-located joins are discussed in detail later.

Do not choose the random distribution policy or a default policy because it is the easy choice. The intent is to distribute the data to reduce movement, which can increase query times. You can introduce issues where one segment may be working harder if your data is skewed.

Use random distribution if an appropriately chosen single or multi-column distribution key does not distribute data evenly across the segments.

## Hash Distributions and Data Skew



**Note:** Select a distribution key with unique values and high cardinality.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

27

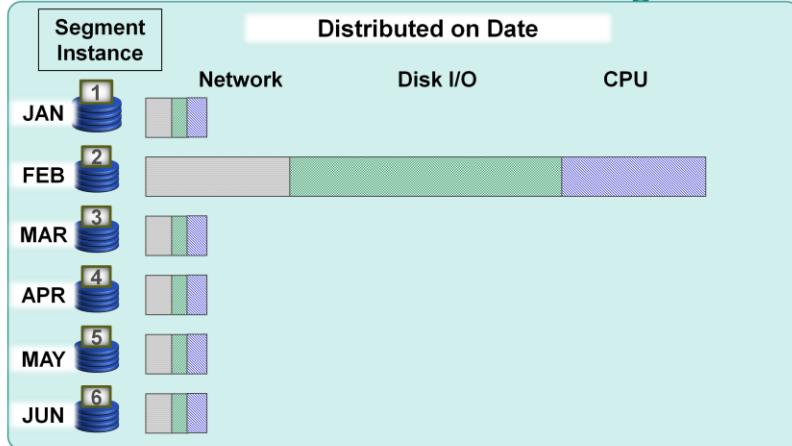
Use a distribution key with unique values and high cardinality to distribute the data evenly across all segment instances.

Boolean keys, such as True/False or 1/0, are not candidates for a distribution key as they will be distributed to two segment instances. Keys with low cardinality, such as Inactive/Active or Male/Female are not candidates for a distribution key.

It is important to remember in an MPP environment, overall response time for a query is measured by the completion time for all segment instances.

In the example displayed a distribution method on a column that contains the value M(ale) and F(emale) is hashed to two hash values. The system distributes rows with the same hash value to the same segment instance therefore resulting in the data being located on only two segment instances.

## Hash Distributions and Processing Skew



Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

28

It is important to select a distribution key to distribute the data evenly across all segment instances but it is equally important to consider processing skew when selecting a distribution key.

For example, using a DATE column as the distribution key may distribute rows evenly across segment instances but may result in processing skew if most of the analysis (queries) is performed by date. Massively parallel processing won't be achieved when all of the records to be processed for a given date are located on a single segment instance.

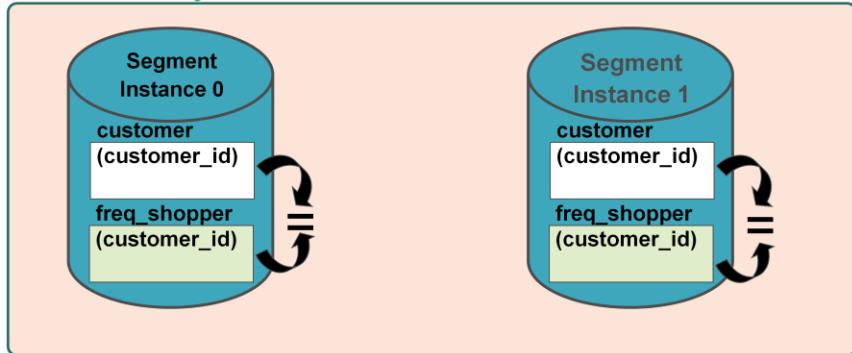
In the example displayed, the DATE column of a table provided an even distribution across all segment instances. Data records are distributed evenly by month with only slight variations in the number of rows in a month. While this resulted in an even distribution, analysis is performed by month with queries analyzing a month of data at a time (SELECT ... FROM table\_name WHERE month = "FEB"). In this example a query WHERE month = "FEB" will result in processing skew with only one segment instance working on behalf of the query.

Remember in an MPP environment, response time for a query is measured by the completion time for all segment instances.

Use iostat to check for processing skew. Use gpssh for ssh access to run system commands on multiple segment hosts at the same time.

If you distribute on a good column where the data is laid out well, but some segments end up working harder than other segments, you introduce processing skew. In such a case, you may be better served distributing on a unique key to distribute the data more efficiently.

## Using the Same Distribution Key for Commonly Joined Tables



**Note:** Optimize for local joins. Distribute on the same key used in the JOIN.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

29

For large tables that are commonly joined together use the same distributed key to obtain local joins. A local join that is performed within the segment instance minimizes data movement and provides tremendous performance gains. Distribute on the same key (column) used in the join (WHERE clause).

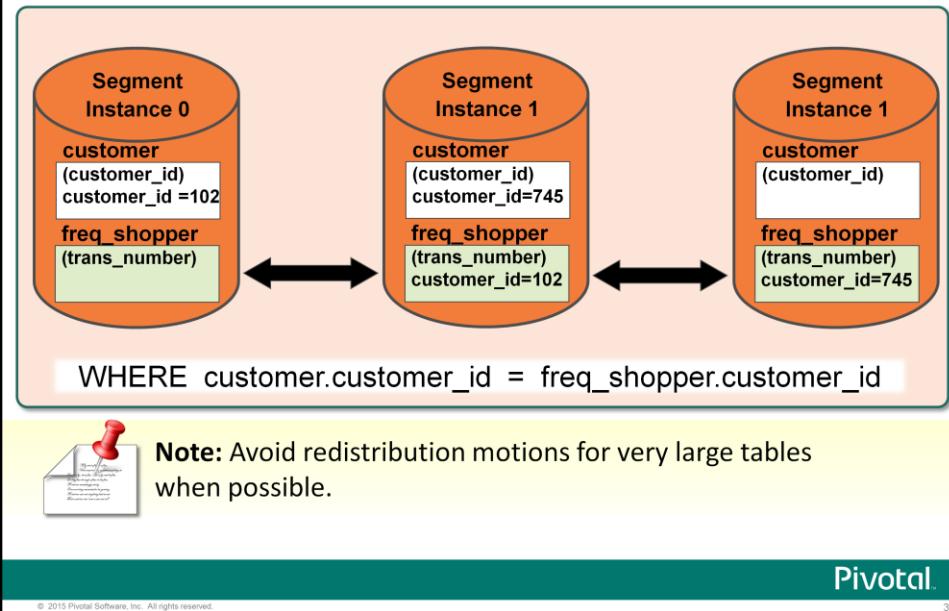
When a local join is performed a segment instance operates independently of the other segment instances without network traffic or communication between segment instances.

In the point-of-sale example displayed the customer and `freq_shopper` table are very large tables and are commonly joined together on the `customer_id` in queries. The `customer_id` is also unique and provides for an even distribution of data. Therefore the same distribution key (`customer_id`) is used for both the `customer` and `freq_shopper` table to obtain the performance benefits of local joins.

To join the `customer` and `freq_shopper` tables each segment instance simply performs the following:

1. Scan the `customer` table that is already distributed on the join key and hashing it.
2. Scan the `freq_shopper` table that is already distributed on the join key and hashing it.
3. Perform a hash join operation between the two tables all locally within the segment instance without network activity.

## Avoid Redistribute Motion for Large Tables



To perform a local join matching rows must be located together on the same segment instance. In the case where data was not distributed on the join key, a dynamic redistribution of the needed rows from one of the tables to another segment instance will be performed.

There is a performance cost to redistributing data that must be performed every time the join is required for a query. Though the performance impact is minimal for small tables, avoid redistribution motion for very large tables when possible.

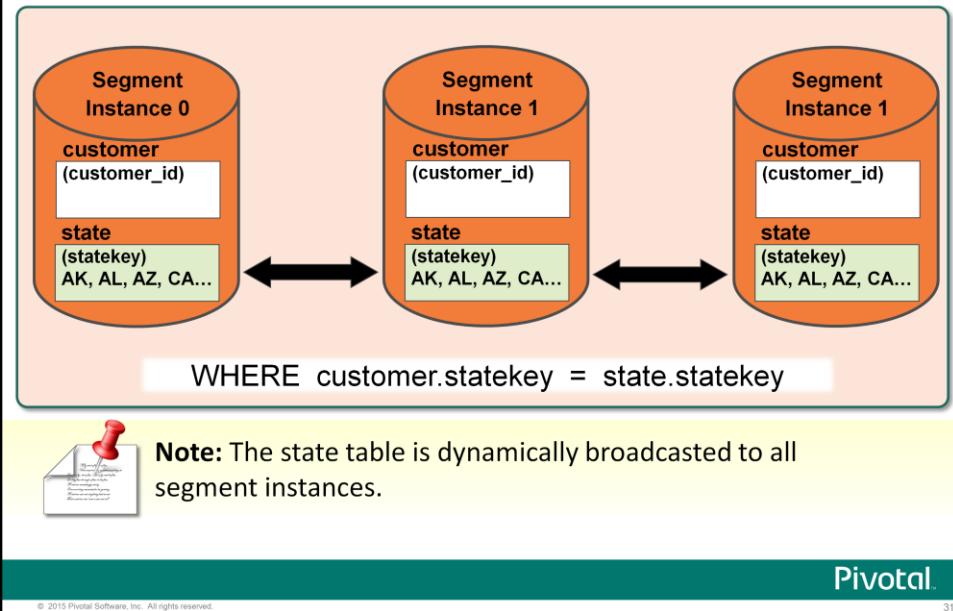
When a redistribution is required the basic steps must still be performed:

1. Scan one table.
2. Scan the second table.
3. Perform the join with an additional step of redistributing the data before it can be joined.

In the example displayed, the `customer` table was distributed on the `customer.customer_id` and the `freq_shopper` table was distributed on `freq_shopper.trans_number`.

To perform the join between the `customer` and `freq_shopper` table WHERE the `customer.customer_id = freq_shopper.customer_id`, the `freq_shopper` table must be dynamically redistributed on the `freq_shopper.customer_id`.

## Avoid Broadcast Motion for Large Tables



In some cases, a broadcast motion will be performed rather than a redistribute motion. In a broadcast motion every segment instance performs a broadcast, or sends its own individual rows to all other segment instances. This results in every segment instance having its own complete and local copy of the entire table.

A broadcast motion may not be as optimal as a redistribute motion therefore the optimizer typically only selects a broadcast motion for very small tables. A broadcast motion is not acceptable for large tables.

In the example displayed, the state table was distributed on the `state_key` and the customer table was distributed on the `customer_id`. It was not feasible to distribute the customer table on the `state_key` because it would result in data skew and the customer table is commonly joined using the `customer_id` with other tables.

The `customer` table is very large and the `state` table is very small relative to the number of rows. The optimizer rather than redistributing the `customer` table decides to broadcast the `state` table.

Each segment instance will send their individual rows from the `state` table to all other segment instances. Each segment instance will then have its own complete and local copy of the `state` table, to join with the `customer` table.

## Commonly Joined Tables Use the Same Data Type for Distribution Keys

Values may appear to be the same, but:

- They are stored differently at the disk level
- They hash to different values
  - Resulting in like rows being stored on different segment instances
  - Requiring a redistribution before the tables can be joined

The following shows two distribution keys with different data types:

```
customer (customer_id)    745::int
freq_shopper (customer_id) 745::varchar(10)
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

32

To perform a local join, matching rows must be located together on the same segment instance. It is important that the distribution keys are the same data type to obtain a local join.

While the values might appear to be the same representatively, different data types are stored differently at the disk level. The data types hash to different values resulting in like rows being stored on different segment instances.

In the example displayed:

- The `customer` table is distributed on the `customer.customer_id` which is defined as type `INT`.
- The `freq_shopper` table also distributed on the `freq_shopper.customer_id` is defined as type `VARCHAR`.
- This results in like rows, with the same `customer_id` respectively, being stored on different segment instances, requiring a redistribution to perform the join.

Instead of a local join, the optimizer redistributes the `customer` table based on the `VARCHAR` representation of the `customer.customer_id`. A hash join is then performed using the same data type for both `customer_id` columns (`customer.customer_id = varchar(freq_shopper.customer_id)`).

## Distributed With DISTRIBUTED RANDOMLY

The DISTRIBUTED RANDOMLY clause:

- Uses a random algorithm
- Requires a redistribute or broadcast motion for any query that joins to a distributed randomly table
- Is acceptable for small tables such as dimension tables
- Is not acceptable for large tables such as fact tables

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

33

The DISTRIBUTED RANDOM clause distributes the data across all segment instances using a random distribution algorithm. For any table that uses a random distribution either a redistribution or broadcast operation will be required to perform the table join.

There are performance implications when performing a redistribution or broadcast of very large tables.

Random distribution should be used for small tables and when a hash distribution method is not feasible due to significant data skew.

## Distribution Strategies to Avoid

Avoid the following when deciding on a distribution table:

- Generate or fabricate columns to create a unique distribution key to avoid skew. For example, do not do this:  
`DISTRIBUTED BY (col1, col2, col3)  
if col1, col2, col3 is never used in joins`
- Creating a fabricated column just to generate a unique key
- Distributing on Boolean data types
- Distributing on floating point data types
- Using `DISTRIBUTE RANDOMLY` because it is the easy choice

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

34

Two of the most influential factors in optimizing the performance in a Greenplum data warehouse include:

- A distribution strategy that distributes rows evenly across all segment instances.
- A distribution strategy that achieves local joins.

It is important to keep in mind it is possible to negatively impact performance through bad distribution decisions and overly complex strategies.

Do not combine columns in a multi-column distribution simply to generate a unique key to avoid data skew if the multi-column distribution key is never used in joins. In this scenario either a redistribution or broadcast operation will be required to perform the table join.

Never create fabricated or concocted columns to a table to generate a unique key (for example a homegrown serial key generation algorithm) to provide for an even distribution.

In both of these scenarios if a good distribution key is not available distribute the data using the random distribution method.

Do not distribute on Boolean data types or floating point data types. Boolean distributions will result in data skew. Floating point data types result in lossy arithmetic and diminished performance as hash join can not be performed on floating point columns. Favor integer column types for distribution columns.

## Check for Data Skew

Check for data skew with the following:

- gp\_toolkit administrative schema offers two views:
  - gp\_toolkit(gp\_skew\_coefficients
  - gp\_toolkit(gp\_skew\_idle\_fractions
- To view the number of rows on each segment, run the following query:

```
SELECT gp_segment_id, count(*)  
FROM table_name GROUP BY gp_segment_id;
```

- Check for processing skew with the following query:

```
SELECT gp_segment_id, count(*) FROM table_name  
WHERE column='value' GROUP BY gp_segment_id;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

35

It is important to validate the table distributions and to check for data skew. Use the gp\_toolkit administrative schema to check for uneven table distributions.

Individual segment host performance may also be viewed during query processing using the Greenplum Command Center.

## Processing and Data Skew Example



### Example: Determining the processing and data skew of existing tables

```
SELECT sifrelnname, siffraction, skccoeff
from gp_skew_idle_fractions, gp_skew_coefficients
WHERE sifrelnname=skcrelname AND siffraction > 0.1;
      sifrelnname |       siffraction |       skccoeff
-----+-----+-----+
car_data | 0.50000000000000000000 | 141.42135623730949000
correlation | 0.40000000000000000000 | 94.280904158206336667000
golfevaluate | 0.22222222222222222222 | 40.406101782088430000000
golfnew | 0.22222222222222222222 | 40.406101782088430000000
golftest | 0.22222222222222222222 | 40.406101782088430000000
old_regr_example | 0.50000000000000000000 | 141.42135623730950000
old_regr_example_test | 0.50000000000000000000 | 141.42135623730951000
sales_example | 0.50000000000000000000 | 141.42135623730951000
water_treatment_predict | 0.12037037037037037 | 19.352396116684458526000
winequality_white | 0.10293040293040293040 | 16.226786893705185790000
```

Pivotal.

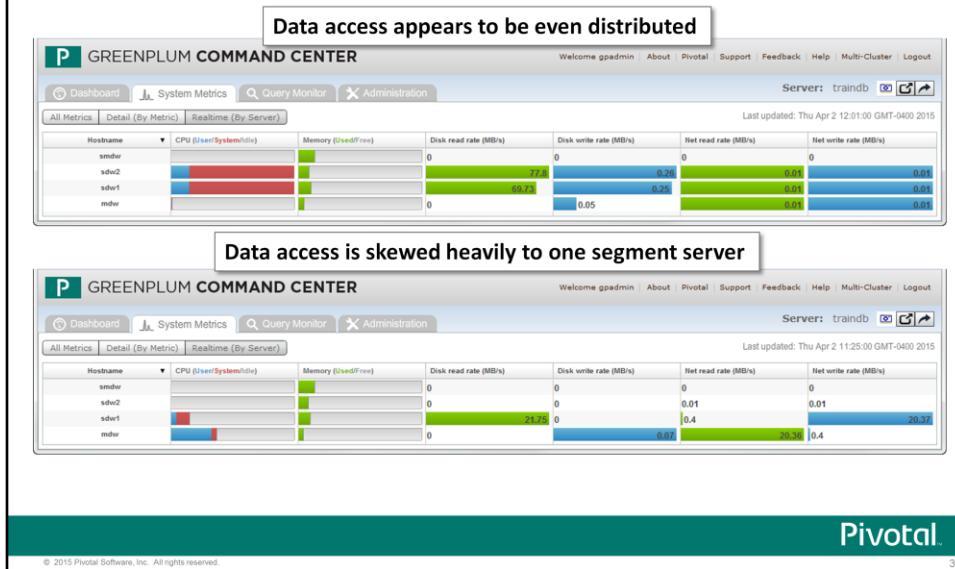
© 2015 Pivotal Software, Inc. All rights reserved.

36

In this example, an SQL query is submitted against the `gp_skew_idle_fractions` view and the `gp_skew_coefficients` view to identify any tables with a processing skew greater than 10%. The related coefficient variable (`skccoeff`) is also displayed.

Note, the greater the processing skew, the greater the coefficient variable.

## Real-Time Data and Processing Skew in Command Center



Greenplum Command Center displays real time data reads and writes, CPU processing, and network reads and writes.

The chart can be used to determine if one or more segments display greater processing or computational skew than others. Data access may change during the execution period of the in-flight SQL statement, so it is important to not look just at a single moment but to watch the update during the entire time the query is being executed.

## Redistribute Using ALTER TABLE

Use the ALTER TABLE command to:

- Redistribute on the current policy

```
ALTER TABLE sales SET WITH (REORGANIZE=TRUE);
```

- Redistribute with a new distribution key

```
ALTER TABLE sales SET DISTRIBUTED BY (column);
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

38

Use the ALTER TABLE SQL command to redistribute a table either on the current distribution policy to rebalance data, or by specifying a new distribution policy to distribute on the newly chosen distribution key.

To redistribute:

- On the current policy, specify the SET WITH (REORGANIZE=TRUE) clause.
- On a different distribution key, use the SET DISTRIBUTION BY (*column*) clause, where *column* is the column name for the new distribution key.

Data will be redistributed automatically and will redistribute recursively on child tables.

## Why Partition a Table?

The following are reasons to partition a table:

- Provide more efficiency in querying a subset of large data
- Increase query efficiency by avoiding full table scans
- Without the overhead and maintenance costs of an index for date
- To handle increasing volumes of data that are not needed by the average query
- Allow instantaneous dropping of older data and simple addition of newer data
- Supports a *rolling n* period methodology for transactional data

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

39

While Greenplum is not afraid of full table scans, it is a good idea to partition your table to help Greenplum process the table as efficiently as possible. However, your business rules should dictate if and how you partition tables.

The following are some of the reasons for considering table partition as part of the design:

- Provide more efficiency in querying against a subset of large volumes of transactional detail data as well as to manage this data more effectively. Businesses have recognized the analytic value of detailed transactions and are storing larger and larger volumes of this data.
- Increase query efficiency by avoiding full table scans.
- Without the overhead and maintenance costs of an index for date.
- As the retention volume of detailed transactions increases, the percent of transactions that the “average” query requires for execution decreases.
- Allow the instantaneous dropping of old data and the simple addition of newer data that may be queried more often. One of the practical uses of partition tables is partition swapping or partition exchanging. This type of practice allows you to load data to a table and replace an existing table with the same data plus the updated information with little impact to the original table.

Support a *rolling n periods* methodology for transactional data.

You can have default partitions that act as a catchall. This allows you to insert rows without having to worry about a partition. Later, you can alter the table to split the partition with the values that you want to go to the partitioned table.

## Candidates for Partitioning

The following date-related tables make excellent candidates for table partitioning:

- Fact tables with dates
- Transaction tables with dates
- Activity tables with dates
- Statement tables with numeric statement periods (YYYYMM)
- Financial summary tables with end of month dates

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

40

Any table with large data volumes where queries commonly select for data using a date range or where the table has roll-off requirements makes an excellent candidate for table partitioning. Most of your partitions will be partitioned on dates. This allows you to implement temperature-sensitive data storage.

Candidates for partitioning include:

- Fact tables with dates
- Transaction tables with dates
- Activity tables with dates
- Statement tables with numeric statement periods
- Financial summary tables with end of month dates

## Partitioning Column Candidates

Column types that can be used as candidates include:

- Dates using date expressions:
  - Trade date
  - Order date
  - Billing date
  - End of Month date
- Numeric period that are integers:
  - Year and month
  - Cycle run date
  - Julian date

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

41

Columns used for partitioning should be simple and meet your business requirements. Examples for dates include:

- Dates with date expressions:
  - Trade date
  - Order date
  - Billing date
  - End of month date
- Numeric representations for dates, including:
  - Year and month
  - Cycle run date
  - Julian date

## Table Partitioning Best Practices

Use table partitioning:

- On large distributed tables to improve query performance.
- If the table can be divided into rather equal parts based on a defining criteria, such as by date.
- If the defining partitioning criteria is the same access pattern used in query predicates, such as WHERE date = '1/30/2008'
- If there are no overlapping ranges or duplicate list values.

Avoid using default partitions in your design

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

42

Table partitioning is not a substitute for table distributions. Therefore, it is important to create optimal data distributions for each table.

The primary goal of table partitioning is to eliminate scanning partitions that contain data that is not needed to satisfy a query.

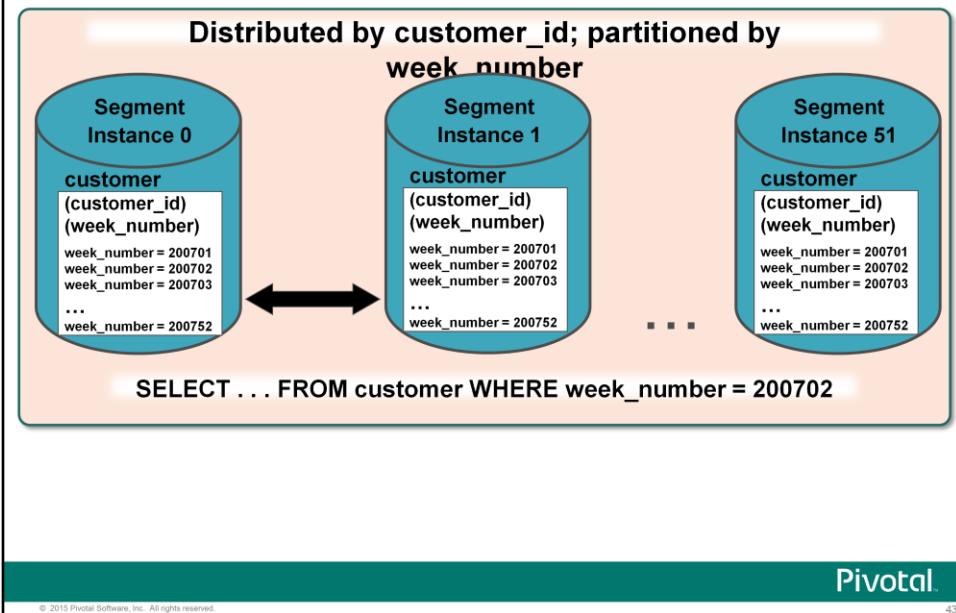
Consider table partitioning on large tables that can be divided into somewhat equal parts based on a defining criteria and the defining criteria is used in query predicates, the WHERE clause. The WHERE clause is your biggest identifier for partitioning.

If the query access pattern (SELECT....WHERE) does not match the partitioning definition the benefit of partition elimination is not achieved.

When defining partitioning criteria it is important to not have overlapping ranges if using range partitioning and to ensure list values are unique if using list partitioning. The query planner will have problems identifying which segment to put the data into.

While default partitions provide a method for catching all data that does not fit other partitions, the use of a default partition can negatively impact performance. Default partitions are always scanned. If the partition contains a large amount of data, this can negatively impact performance during table scans.

## Table Partitioning Example



In the example displayed on the slide, the `customer` table was distributed on `customer_id` providing for an even distribution across all segment instances.

In this data warehouse environment example, a large percentage of the queries are against a small subset of history. For example:

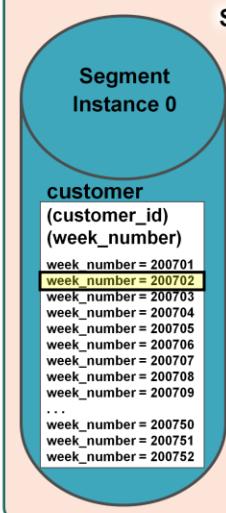
- One week (`WHERE week_number = 200702`)
- One month (`WHERE week_number = 200701 AND week_number = 200702 AND week_number = 200703 AND week_number = 200704`)
- One quarter

In this scenario, query performance can be improved by eliminating partitions to be scanned that contain the other weeks that are not needed to satisfy the query.

It was also determined that `week_number` can be easily divided into equal parts with no overlapping ranges. Therefore the `customer` table is partitioned on `week_number` with each segment instance containing fifty-two partitions or child tables based on week number.

## Partition Elimination

SELECT . . . FROM customer WHERE week\_number = 200702



Only the 200702 week partition is scanned.

The other 51 partitions on the segment instance are eliminated from the query execution plan.

**Note:** The primary goal of table partitioning is to achieve partition elimination.

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

44

Partition elimination is the act of filtering and eliminating data irrelevant to a query. By reducing table scans, query processing significantly improves, allowing for more concurrency.

For a query that selects a single week, for example WHERE week\_number = 200702, the remaining fifty-one partitions within the segment instance are eliminated from the query execution plan and are not scanned. This results in improved query performance.

The overall goal of table partitioning is to achieve partition elimination to improve query performance. This can occur either as part of the query plan or during query run-time.

## Dynamic Partition Elimination

### Example: Build a partition table using list values

```
CREATE TABLE factotperf_quarter (LIKE factontimeperformance)
DISTRIBUTED BY(airlineid)
PARTITION BY LIST(quarterid)
(PARTITION first_quarter VALUES(1),
PARTITION second_quarter VALUES(2),
PARTITION third_quarter VALUES(3),
PARTITION fourth_quarter values(4));
```

### Example: Query a fact and dimension table to trigger dynamic partition elimination

```
SELECT *
FROM factotperf_quarter, dimquarter
WHERE dimquarter.quarterdescription like 'Quarter1%' AND
factotperf_quarter.quarterid=dimquarter.quarterid;
```

Partitions are eliminated at run time **before** a join between the two tables occur. A filter is applied to each partition to activate those that match the criteria specified.

Restriction is not on the partition key

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

45

Partition elimination that occurs during run-time is called dynamic partition elimination. Dynamic partition elimination is used whenever the values are only available at query run-time. These values are used to prune partitions dynamically, once again, improving the speed of query processing.

This can occur whenever the restriction defined as part of the WHERE clause is not on the partitioning key. While the feature was available in earlier releases of Greenplum 4.x, it applied to constant values known to the partition table at planning time. The feature was extended to eliminate partitions at run-time.

Dynamic partition elimination is on by default, but can be disabled by setting the server configuration parameter, gp\_dynamic\_partition\_pruning, to off.

## Creating Partitioned Tables

The following shows how to create the customer partition table using an interval:



Example: Create a partition table using an interval

```
CREATE TABLE customer (
    customer_id      INT,
    week_number      INT,
    ...
) DISTRIBUTED BY (customer_id)
PARTITION BY RANGE ( week_number )
(START (200701) END (200752) INCLUSIVE EVERY (INTERVAL '1
WEEK'));
```

Pivotal

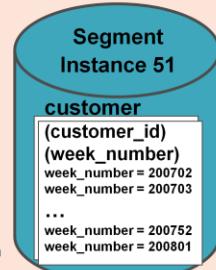
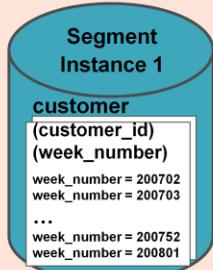
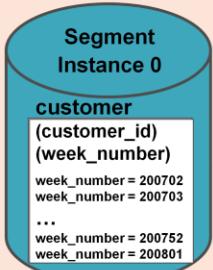
© 2015 Pivotal Software, Inc. All rights reserved.

Once a partition strategy is determined that provides for partition elimination, the partitions or child tables must be created.

Use the `CREATE TABLE` statement with the `INHERITS` clause to create the parent-child relationship and inheritance from the parent table. Use the `CHECK` constraints clause to limit the data a child table can contain based on some defining criteria.

## Maintaining Rolling Windows

Use ALTER TABLE with ADD PARTITION clause to add a new child table



Use ALTER TABLE with DROP PARTITION clause to delete an old child table

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

47

Many data warehouse environments may contain rolling history data that represents many weeks, months or years of activity. A rolling history table typically contains a very large number of records. For these tables a rolling window of data is stored and maintained based on a defining criteria.

Continuing with the example:

- Fifty-two weeks of data are stored and analyzed in the customer table.
- Each week, a new week's worth of data is inserted and the oldest week's worth of data is deleted, maintaining fifty-two rolling weeks of data at any given time.
- As these rows are historical in nature, they are rarely, if ever, modified after insertion.

To maintain the rolling window of data:

- Use ALTER TABLE command with the ADD PARTITION clause to a new partition or subpartition containing the new data.
- Use ALTER TABLE command with the DROP PARTITION clause to remove a partition containing older data.

## Partition Table Design

Example: Create a partition table based on business requirements

```
CREATE TABLE orders
  ( order_id          NUMBER(12),
    order_date        TIMESTAMP WITH LOCAL TIME ZONE,
    order_mode        VARCHAR2(8),
    customer_id       NUMBER(6),
    ...
    promotion_id      NUMBER(6)
  )
DISTRIBUTED BY (customer_id)
PARTITION BY RANGE (order_date)
(
  START (date '2005-12-01')
  END (date '2007-12-01') EVERY '3 months',
  END (date '2008-12-01') EVERY '4 weeks',
  END (date '2008-12-03') EVERY '6 hours',
  END (date '2008-12-04') EVERY '2 secs'
);
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

48

The business requirements dictate the design you choose when partitioning a table. In this example, the SQL syntax to create a single-level partition for the `orders` table, where there no partitions are defined, is displayed.

The table is partitioned by a range on the `order_date` column, with data being created:

- Every 3 months for a specific date range
- Every 4 weeks for another date range
- Every 6 hours for a third date range
- Every 2 seconds for the last date range shown

## Data Type Best Practices

The following should be considered when defining columnar data types:

- Use data types to constrain your column data:
  - Use character types to store strings
  - Use date or timestamp types to store dates
  - Use numeric types to store numbers
- Choose the type that uses the least space:
  - Do not use a BIGINT when an INTEGER will work
  - Use identical data types for columns used in join operations
  - Converting data types prior to joining is resource intensive

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

49

Use the correct data types to store your values. Verify that the data types match on JOIN operations. You must analyze your table design to ensure they can be matched on joins. Joins work more efficiently if the data types of the columns used in the join predicate have identical data types. When the data types differ, the database must convert one of them so that the data types can be compared correctly, amounting to unnecessary overhead.

## Table and Column Constraints

The following code creates:

- A table with a CHECK constraint:

```
CREATE TABLE products
( product_no integer, name text,
  price numeric CHECK (price > 0) );
```

- A table with a NOT NULL constraint:

```
CREATE TABLE products
( product_no integer NOT NULL, name text NOT NULL,
  price numeric );
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

50

Constraints give you more control over the data in table, where data types may not provide enough of control over the data defined. If a user tries to enter data that violates the constraint you defined for a column, an error is raised.

The slide shows how to create supported types of constraints on the table and columns:

- In the first example, a CHECK constraint is placed on the price column of the products table. This ensures that for any price values entered must be greater than 0.
- The second example shows a NOT NULL constraint, where the product\_no and name columns cannot be NULL or empty.

## Table and Column Constraints (Cont)

- A table with a UNIQUE constraint:

```
CREATE TABLE products
( product_no integer UNIQUE, name text, price numeric)
DISTRIBUTED BY (product_no);
```

- A table with a PRIMARY KEY constraint

```
CREATE TABLE products
( product_no integer PRIMARY KEY, name text, price
numeric)
DISTRIBUTED BY (product_no);
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

51

- The first example on this slide shows a UNIQUE constraint on the product\_no column, where each value is distinct. The table must be hash distributed and cannot be distributed randomly. The constraint column must either be the same, or a superset, of the table's distribution key columns.
- The second example on this slide shows the PRIMARY KEY constraint on the product\_no column. The PRIMARY KEY constraint enforces UNIQUE and NOT NULL constraints on the column. The table must be hash distributed and cannot be distributed randomly. The constraint column must either be the same, or a superset, of the table's distribution key columns.

## Lab: Physical Design Decisions

In this lab, you create table objects based on business requirements provided. You use a combination of constraints to maintain control of data on the tables.

You will:

- Create the Store dimension
- Create the Country dimension
- Create the Customer dimension
- Create the Transaction fact table
- Load dimension and fact data

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

52

In this lab, you create table objects based on business requirements provided. You use a combination of constraints to maintain control of data on the tables.

# Module 7: Data Modeling and Design

## Lesson 2: Summary

During this lesson the following topics were covered:

- Selecting the best distribution key used for distributing data
- Check for data skew
- The benefits of partitioning a table and when to partition a table
- Determining partitioning candidates
- Selecting appropriate data types for your data
- Defining constraints on tables and columns

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

53

This lesson discussed how to design and define your tables to best suit the needs of its users, including selecting the best distribution key for distributed data. Distribution keys impact how the data is distributed across segments and therefore can impact scan rate times, particularly if data is skewed. Steps to check for skew are provided in the lesson.

When to partition a table and why you should do so is also reviewed in this lesson. Selecting ideal partitioning candidates is based on how your users access the data.

Selecting appropriate data types can impact physical storage size as well as join performance for queries, so data types should also be a consideration when defining tables and fields.

Lastly, defining constraints on tables and columns can help you to control the type of data within a table or partition and should be considered when your data needs to follow strict rules.

## Module 7: Summary

Key points covered in this module:

- Identified and described the data models used in data warehousing and describe how data is stored in Greenplum
- Distributed and stored data in Greenplum using a distribution key, partitioning, and constraints

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

54

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

- Identify and describe the data models used in data warehousing and describe how data is stored in Greenplum.
- Distribute and store data in Greenplum using a distribution key, partitioning, and constraints.

# Module 8: Performance Analysis and Tuning

This module more closely examines performance analysis and tuning using advanced SQL concepts.

Upon completion of this module, you should be able to:

- Combine data from multiple tables using JOINs
- Use EXPLAIN and EXPLAIN ANALYZE to help the optimizer determine how to handle a submitted query
- Improve query performance by keeping statistics up to date and tuning the database for sampling size and error conditions
- Determine when it is best to use an index and what type of index to use

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

1

This module approaches performance analysis and tuning through advanced SQL topics. You learn about joining tables and the impact that different types of joins can have on overall performance. Statistics collection and analysis continue to play a key role in reducing the time the query planner and the query perform. Indexes are not always beneficial to your data, so we examine specifically how they can be used. OLAP groups and windowing functions offers benefits not only for coding, but also to improve performance.

In this module, you will:

- Combine data from multiple tables using JOINs.
- Use EXPLAIN and EXPLAIN ANALYZE to help the optimizer determine how to handle a submitted query.
- Improve query performance by keeping statistics up to date and tuning the database for sampling size and error conditions.
- Determine when it is best to use an index and what type of index to use.

# Module 8: Performance Analysis and Tuning

## Lesson 1: JOIN Tables – Types and Methods

In this lesson, you examine join types you employ in combining data from multiple tables, the join methods Greenplum uses for query plans, and potential impact to performance of each.

Upon completion of this lesson, you should be able to:

- List the different types of joins that can be performed on tables
- Describe how row elimination can be used to improve performance in query execution
- Identify how to minimize data movement during joins with the use of the distribution key
- List join methods commonly seen in query plans and the potential performance impact of each

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

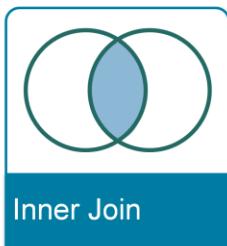
2

When data is separated across multiple tables, joins are a common way of combining the data from two or more database tables to create a single set of data. Maintaining data in different table, such as when using the 3NF data model, can make it easier to maintain, but it can have performance impacts when querying against that data to return meaningful subsets of data.

In this lesson, you will:

- List the different types of joins that can be performed on tables.
- Describe how row elimination can be used to improve performance in query execution.
- Identify how to minimize data movement during joins with the use of the distribution key.
- List join methods commonly seen in query plans and the potential performance impact of each.

## JOIN Types



Inner Join



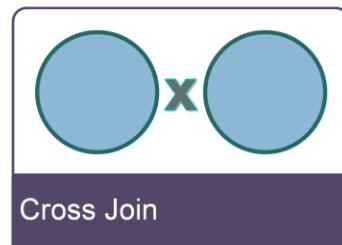
Left Outer Join



Right Outer Join



Full Outer Join



Cross Join

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

3

The five major types of joins that will be discussed in this lesson include:

- **Inner join** – The inner join is possibly the most common type of join. The resulting data set is obtained by combining two tables on a common column. Each row of the left table is compared against each row of the right table. All matching rows are returned as part of the result set. An equijoin is an inner join that uses only equality comparisons in the join predicate.
- **Left outer join** – Left outer join returns all of the rows from the left table even if there is no matching row in the right table. It also returns matching rows from the right table. Rows in the right table that do not match are not included as part of the result set.
- **Right outer join** – Right outer join returns all of the rows from the right table even if there is no matching row in the left table. It also returns the matching rows from the left table.
- **Full outer join** – Full outer join returns all rows from both tables where there is a match and returns NULL for rows that do not have a match.
- **Cross join** – Cross join returns the Cartesian product of rows from tables in the join. The resulting data set consists of a combination of each row in the left table with each row in the right table. Two tables, each with five rows, will produce a resulting data set that contains twenty-five rows.

## Inner Join

Inner joins:

- Can be simple to write
- Require a join condition be specified
- Are also known as a SIMPLE JOIN or EQUI-JOIN



The following is an example of an inner join:

Example: Join two tables using an equijoin

```
SELECT
    EMP.EMPLOYEE_ID
    , EMP.EMPLOYEE_NAME
    , DPT.DEPT_NAME
FROM
    EMPLOYEES    EMP
    , DEPARTMENTS DPT
WHERE
    EMP.DPT_ID = DPT.DPT_ID;
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

4

The syntax for an inner join is simple to develop. It does require that a join condition be specified – one cannot be implied. It is also known as a simple join or an equi-join.

In the inner join example shown, the DPT\_ID columns of the EMP and DPT table are compared to each other. This query only retrieves rows based on the following join condition: A corresponding row must exist in each table, thus the term INNER JOIN.

## ANSI Syntax for Joins



### Example: Join two tables using an inner join

```
SELECT
    EMP.EMPLOYEE_ID
    , EMP.EMPLOYEE_NAME
    , DPT.DEPT_NAME
FROM
    EMPLOYEES      EMP
    INNER JOIN DEPARTMENTS DPT
    ON ( EMP.DPT_ID = DPT.DPT_ID );
```



### Example: Alternative usage of INNER without the JOIN

```
...
FROM
    EMPLOYEES      EMP
    JOIN DEPARTMENTS DPT
    ON EMP.DPT_ID = DPT.DPT_ID;
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

5

Both examples on the slide are alternative methods for writing an inner join. The first syntax uses the term `INNER JOIN` between the two tables followed by the predicate `ON (table1.column = table2.column)`.

The second example uses `JOIN` between the two tables with the same `ON` predicate.

Note that you must use ANSI syntax for any join other than `INNER`.

## Left Outer Join

A left outer join:

- Returns all rows from the left table and rows from the right table that match the left one
- Is used when the right table can supply information missing from the left table
- Can be referenced as



 Example: LEFT OUTER JOIN example

```
SELECT
    t.transid
    , c.custname
FROM
    facts.transaction t
    LEFT OUTER JOIN dimensions.customer c
    ON c.customerid = t.customerid;
```

© 2015 Pivotal Software, Inc. All rights reserved.

Pivotal.

6

### LEFT OUTER JOIN:

- Returns all rows from the left table and only those rows from the right table that match the left one. This is handy when there may be missing values in the left table, but you want to get a description or other information from the right table.
- Is defined with the syntax `LEFT OUTER JOIN` or abbreviated to `LEFT JOIN`.
- Extends the rows from the left table to the full width of the joined table by inserting null values for the right-hand columns.

## Right Outer Join

A right outer join:

- Is the inverse of the LEFT OUTER JOIN
- Is functionally the same as the LEFT OUTER JOIN, but care must be taken when specifying the left and right tables
- Can be referenced as RIGHT OUTER JOIN or RIGHT JOIN



### Example: RIGHT OUTER JOIN example

```
SELECT
    t.transid
    , c.custname
FROM
    dimensions.customer c
RIGHT OUTER JOIN facts.transaction t
ON c.customerid = t.customerid;
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

7

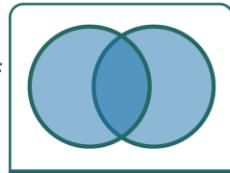
A right outer join:

- Returns all the joined rows, plus one row for each unmatched right-hand row.
- Is functionally the same as the LEFT OUTER JOIN. However, in either case, you must take care to choose the left table and the right table.
- Can be referenced either as RIGHT OUTER JOIN or RIGHT JOIN.

## Full Outer Join

Full outer join:

- Is used to retrieve every row irrespective of match
- Can be used to detect *true changes*



Example: FULL OUTER JOIN example

Full Outer Join

```
SELECT COALESCE( ORIG.transid, STG.transid) AS transid,
CASE
    WHEN ORIG.transid IS NULL AND STG.transid IS NOT NULL
        THEN 'INSERT'
    WHEN ORIG.transid IS NOT NULL AND STG.transid IS NOT NULL
        THEN 'UPDATE'
        WHEN ORIG.transid IS NOT NULL AND STG.transid IS NULL
            THEN 'DELETE'
        ELSE 'UNKNOWN'
    END AS Process_type
FROM facts.transaction          ORIG
    FULL OUTER JOIN staging.transaction_w   STG
    ON STG.transid = ORIG.transid;
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

8

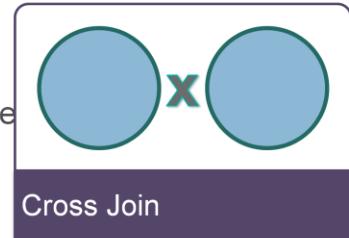
This interesting join is used when you want to retrieve every row, whether there is a match or not based on the join criteria. This type of join can be used for detecting *true changes* for a table.

In the example shown, the COALESCE function ensures that a transaction ID is retrieved to work with in subsequent processing. The FULL OUTER JOIN syntax joins on the transid column that the COALESCE function is performing a check on.

## Cross Join (Cartesian Product)

A cross join:

- Oftentimes is used unintentionally
- Combines every row in the left table with every row in the right table
- Is specified with the CROSS JOIN syntax or by comma separated tables with no predicates



### Example: CROSS JOIN example

```
SELECT
    t.transid,
    t.transdate,
    COALESCE( c.custname, 'Unknown Customer') AS custname
FROM facts.transaction t, dimensions.customer c;
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

9

A cross join may not be the type of join you should use when working on large data. This can result in huge transactions.

For example, if you have a billion row table and CROSS JOIN it to a 100 row table, your resulting set will have 100 billion rows.

It can take much longer to perform a cross join when working with large data. For example, performing a 23 trillion row cross join in one table could take hours to complete the transaction. However, selecting against the table did eventually yield the desired result.

The syntax shown defaults to a cross join, so care should be taken when using this type of syntax. Cross joins can be specified with the CROSS JOIN syntax.

## Example of a Useful Cross Join



### Example: CROSS JOIN example

```
SELECT
    BATCH_USER_ID
    , C.DATE
    , CASE WHEN
        C.DATE >= P.PROJECT_START_DT
    THEN
        SUM( DAILY_HOURS_FORECAST)
        OVER( PARTITION BY P.BATCH_USER_ID, C.DATE)
    ELSE 0
    END AS DAILY_HOURS_FORECAST

FROM
    METADATA.COMPANY_PROJECTS      AS P
    CROSS JOIN PUBLIC.CALENDAR     AS C
WHERE
    P.BATCH_USER_ID IS NOT NULL
    AND P.PROJECT_ACTIVE_FLG = 'N'
    AND PROJECT_START_DT >= '2008-01-01'
    AND C.DATE BETWEEN '2008-01-01' AND CURRENT_DATE;
```

Pivotal

The example shown on the slide is an example of real use of the CROSS JOIN. The code aggregates all of the forecasts for every project, every day, from the Jan 1st, 2008 until today.

## Row Elimination

During join executions:

- Greenplum often has to use disk space to temporarily store data.
- Query optimizer minimizes the amount of memory and disk space required by:
  - Projecting (copying) only those columns that the query requires
  - Doing single-table set selections first (qualifying rows)
  - Putting only the smaller table into the spool whenever possible



**Note:** Non-equality join operators produce a partial Cartesian product. Join operators should always be equality conditions.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

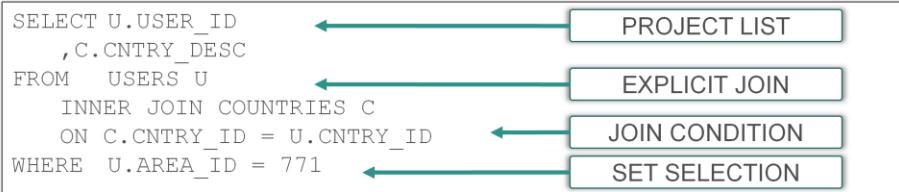
11

When Greenplum executes a plan, it attempts to eliminate as many rows as possible. The optimizer tries to be as restrictive as possible, before the joins, to reduce the number of rows returned for the join. The intention is that the join is performed only on rows and columns needed for the join.

## Analyzing Row Elimination

Row elimination:

- Copies selected rows into temporary tables or spill files
- Projects needed columns into spill file
- Restricts data starting from the WHERE clause, then to the FROM statement, and finally to the SELECT statement



- Small temp space from large tables with few rows and columns
- Large temp space from small tables with many rows and columns

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

12

The example on the slide shows four columns the query is working on:

- The first restriction is developed using the WHERE clause. It restricts the number of rows returned.
- Next, the columns specified in the FROM statement determine which columns are required by the query.
- Finally, the columns selected are specified in the SELECT statement.

Note that you must make sure your data is as clean as possible before the optimizer is invoked as part of the query plan. If the data is not cleaned, vacuum and analyze the tables. This removes any old or stale data, as would occur during high read/write transactions, such as when deleting or updating rows in a table.

In the example shown:

- Row selection is performed as soon as possible. Only rows from the USERS table with a AREA\_ID of 771 are qualified into a temporary table or spill file.
- The Optimizer will typically *project* only the columns needed into a spill file. In this example, the columns, USER\_ID, CNTRY\_ID, CNTRY\_DESC are projected.

The temporary space created for row elimination has the following uses:

- Small temporary space from large tables with few rows or columns projected.
- Large temporary space from small tables with many rows or columns projected.

## Row Redistribution/Motion

During join operations:

- The optimizer uses the distribution key to decide which rows to move.
- Three general scenarios may occur during merge joins:
  - The join column is the distribution key for both tables
  - The join column is the distribution key of one of the tables
  - The join column is the distribution key of neither table.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

13

Data is moved around in spool, temporarily, during redistribution. The optimizer considers how to move data based on the distribution key. There are three general considerations when merging rows:

- The join column is the distribution key for both tables.
- The join column is the distribution key of one of the tables.
- The join column is not the distribution key for either table.

## Row Redistribution – Distribution Key on Both Tables

When the join column is the distribution key of both tables:

- Joinable rows are already on the same target data segment
- No movement of data to other segments is required

JOIN columns are from the same domain.  
No Redistribution needed.

```
SELECT ...  
FROM T1, T2  
WHERE T1.A = T2.A
```

T1			T2		
A	B	C	A	B	C
UPI			UPI		
100	214	433	100	725	2

Pivotal

When the join column is the distribution key on both tables:

- Joinable rows are already on the same target data segment, since equal distribution key values always hash to the same data segment.
- No movement of data to other segments is required.
- The rows are already sorted in hash sequence because of the way in which they are stored by the file system.
- With no need to sort or move data, the join can take place immediately.

## Row Redistribution – Distribution Key on One of the Tables

When the join column is the distribution key of one of the tables:

- One table has its rows on the target segment and one does not
  - Rows of the second table must be redistributed to their target segments

T3			T4		
A	B	C	A	B	C
UPI			UPI		
255	345	225	867	255	566
SPOOL			SPOOL		
A	B	C	A	B	C
	PI				
			867	255	566

When the join column is the distribution key for one of the tables:

- One table has its rows on the target segment and one does not.
  - The rows of the second table must be redistributed to their target segments by the hash code of the join column.
  - If the table is small, the optimizer may decide to simply duplicate the entire table on all segments instead of hash redistributing the data.
  - The rows of one table will be copied to their target data segments.

## Row Redistribution – Distribution Key on Neither of the Tables

When the join column is not the distribution key for either of the tables:

- Both tables may require preparation for joining
- Various combinations of hash distributions or table duplications may be used
- This approach involves the maximum amount of data movement

The screenshot shows a database interface with two tables, T5 and T6, and their corresponding SPOOL outputs.

**T5 Data:**

A	B	C
UPI		
456	777	876

**T6 Data:**

A	B	C
UPI		
993	228	777

**Redistribution Instructions:**

- Redistribute T5 rows on B.
- Redistribute T6 rows on C.

**SQL Query:**

```
SELECT ...  
FROM T5, T6  
WHERE T5.B = T6.C
```

**SPOOL Outputs:**

**T5 SPOOL:**

A	B	C
	PI	
456	777	876

**T6 SPOOL:**

A	B	C
		PI
993	228	777

**Pivotal Logo:**

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

16

When the join column is not the distribution key for either tables in the join:

- Both tables may require preparation for joining.
- This may involve various combinations of hash distributions or table duplications.
- This approach involves the maximum amount of data movement.

## Join Methods

Join methods:

- Are the means that Greenplum database uses to join two tables
- Include:
  - Sort merge join
  - Hash join
  - Nested loop join
  - Product join

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

17

These are the join methods seen in a query plan:

- Sort Merge Join
- Hash Join
- Nested Loop Join
- Product Join

Code designers and developers should question the use of nested loop and product joins in their queries.

Nested loop joins have the potential to run for a very long period of time.

## Sort Merge Join

Sort merge joins:

- Are commonly performed when the join condition is based on equality
- Consists of the following steps:
  - Identifying the smaller table
  - Puts the qualifying data from one or both tables into a spill file, if necessary
  - Moves, or co-locates, the spool rows to the segments based on the join column hash, if necessary
  - Sorts the remaining rows by the join column row hash value, if necessary
  - Compare those rows with matching join column row hash values

Pivotal

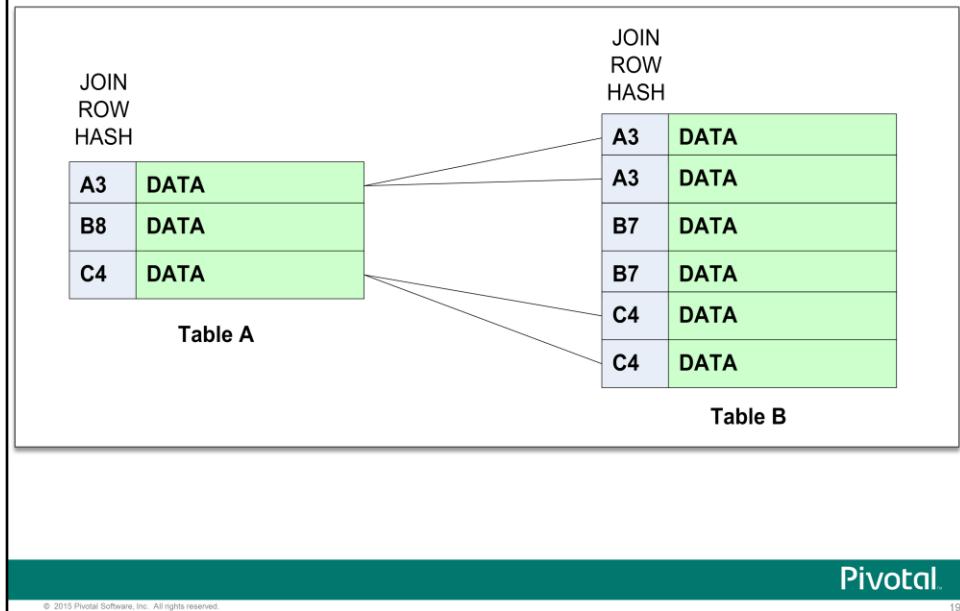
© 2015 Pivotal Software, Inc. All rights reserved.

18

Sort merge joins are commonly performed when the join condition is based on equality. It consists of several steps, the main ones involving identifying the smaller table and comparing its rows against the other table's rows using the join column hash values.

Placing data in the temporary space, moving rows to other segments, and sorting have performance impacts that can slow down a query significantly. These steps are only performed if necessary.

## Illustration of the Sort Merge Join Process



In this illustration, the smaller table, table A, is identified by the optimizer during the join. Values are compared from the smaller table to the larger table. The values in the join row that are matched include A3 and C4. This occurs several times in the larger table, table B.

## Hash Join

In a hash join:

- The smaller table is sorted into join column hash sequence
- The smaller table is duplicated on all data segments
- The larger table is processed one row at a time

The optimizer:

- Can choose this join plan when qualifying rows of the smaller table can be held segment memory resident
- Should have access to the latest statistics

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

20

Merge join requires that both sides of the join have their qualifying rows in join columns hash sequence. In addition, if the join columns are not the primary index columns, some redistribution or duplication of rows precedes the sort.

In a Hash Join, the smaller table is sorted into join column hash sequence and then duplicated on all data segments. The larger table is then processed one row at a time. For those rows that qualify for joining the row hash value of the join columns is used to do a binary search through the smaller table for a match.

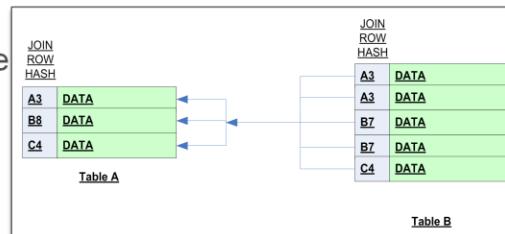
The optimizer can choose this join plan when qualifying rows of the smaller table can be held segment memory resident – the whole row set held in the each segment's memory.

Use ANALYZE on both tables to help guide the optimizer.

## Illustration of the Hash Join Process

The hash join process:

- Identifies the smaller table
- Duplicates it on every data segment
- Sorts into the join column hash sequence
- Holds the rows in memory.
- Uses the join column hash value of the larger table to search memory for a match
- Provides performance benefit in that the larger table does not need to be sorted before the join



Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

21

The hash join process:

- Identifies the smaller table, in this example, A.
- Duplicates the smaller table on every data segment.
- Sorts the rows using the join column hash sequence
- Holds the rows in memory.
- Uses the join column hash value of the larger table to search memory for a match

A performance benefit of this join can result from the fact that there is no need for sorting larger join tables before performing the join.

## Nested Loop Join

A loop join:

- Can be one of the most efficient types of joins.
- Requires the following conditions when joining on tables:
  - An equality value for the unique index of the first table
  - A join on a column between the first table and an indexed column on the second table

In the nested loop join process:

- The system retrieves rows from the first table based on the index value
- The hash value in the join column to access matching rows in the second table is determined
- Not all of the segments may be used

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

22

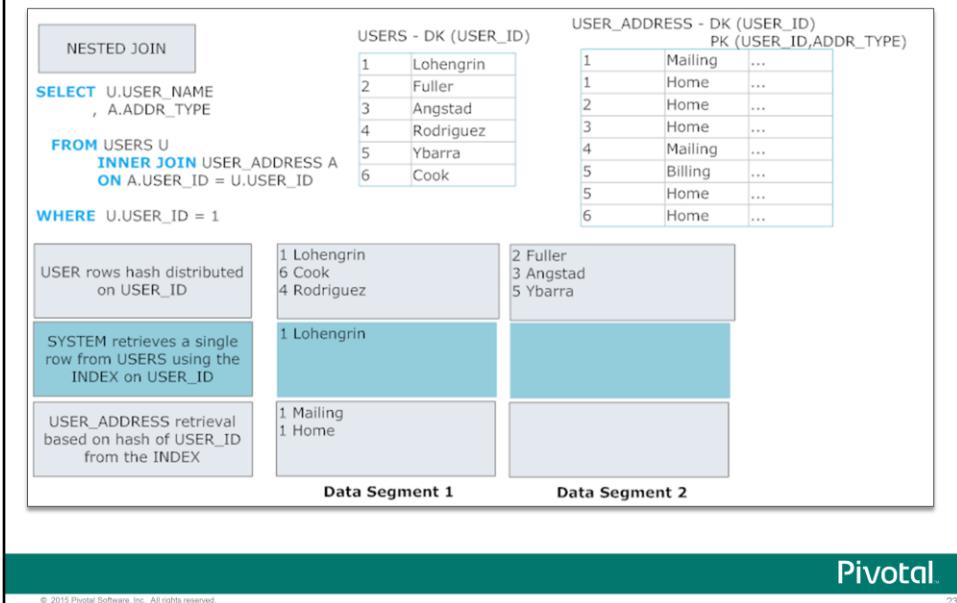
A loop join can be one of the most efficient types of joins. For a nested loop join to be performed between two tables, the following two conditions must occur:

- An equality value for the unique index of the first table. This is retrieved from a single row, or a small number of rows, such as from an IN expression.
- A join on a column between the first table and an indexed column the second table.

During the nested loop join process:

- The system retrieves rows from the first table based on the index value. It then determines the hash value in the join column to access matching rows in the second table.
- Nested loop joins are the only types of Join that do not always use all of the segments during the join process.

## Illustration of the Nested Loop Join Process



The illustration shows the equality value is chosen for the first and smaller table, USERS. The system then retrieves a single row from USERS and joins on an indexed column in the second table, USER\_ADDRESS.

## Product Join

In a product join:

- Every qualifying row of one table is compared to every qualifying row in the second table
- The required number of comparisons is a product of the number of qualifying rows from both tables.
- All rows of one side must be compared with all rows of the other, causing the smaller table to be duplicated on segments
- The rows of the smaller table are compared to the local rows of the larger table

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

24

In a product join, every row in one table will be matched against every row in another table. So if you have 100 rows in one table and 1000 rows in the second table, you'll have 100,000 rows in the end.

The smaller table is always broadcast to the segments. With regards to resources, this can be dangerous. If you have 1 million rows being broadcast, you'll need appropriate storage for holding that amount of data on each segment.

## Conditions that Invoke Product Joins

Product Joins may be caused by any of the following:

- A missing WHERE clause
- A join condition not based on equality, such as not equals, or less than
- Join conditions connected using OR
- Too few join conditions in the WHERE clause
- A referenced table not named in any join condition
- The optimizer determines that it is less expensive than any other join type
- The optimizer determines that it will lead to less expensive joins later in the plan such that the overall plan is less expensive

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

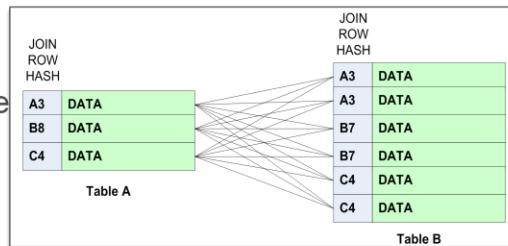
25

In general, product joins are considered to be bad. It is rare when the optimizer chooses product joins over other joins. The optimizer may choose a join when the product join is more expensive but will lead to less expensive joins later in the plan. This choice means that the overall plan is less expensive.

## Illustration of the Product Join Process

The product join process:

- Identifies the smaller table
- Duplicates the smaller table in temporary space or memory on all data segments
- Joins each row of the smaller table to every row in the larger table
- Calculates the number of comparisons as the number of qualified rows in the first table to the number of qualified rows in the second table
- Can have costly comparisons when there are more rows than segment memory can hold at one time



Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

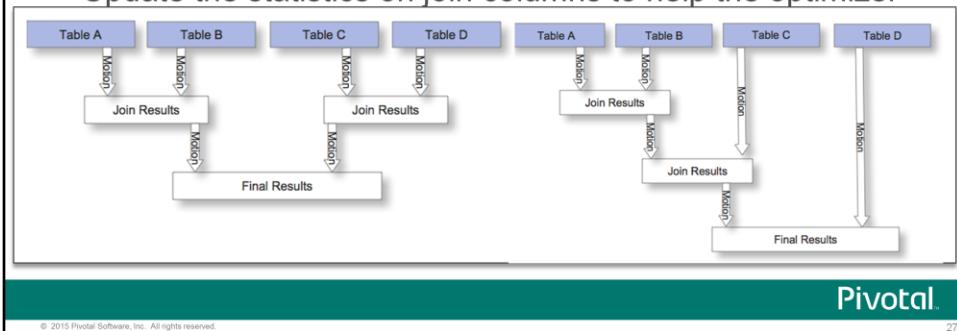
26

In the product join process, the smaller row is duplicated to all data segments and the rows of the smaller table are joined to every row in the larger table. A product join can become quite costly over the course of the process, especially if there are more rows than segment memory can hold at any one time.

## N-Tables

When joining tables:

- All  $n$ -table joins are reduced to a series of two-table joins
- The optimizer attempts to determine the best join order
- The query engine can only work on two tables at a time
- The results of the previous join operation are applied to another table or another join results
- Update the statistics on join columns to help the optimizer



Every join occurs two-tables at a time. It can join using the first method depicted on the slide, or using the second method on the slide. The optimizer uses the statistics retrieved from `ANALYZE` to determine which to use.

# Module 8: Performance Analysis and Tuning

## Lesson 1: Summary

During this lesson the following topics were covered:

- Different types of joins that can be performed on tables
- Row elimination can be used to improve performance in query execution
- Minimize data movement during joins with the use of the distribution key
- Commonly seen join methods in query plans and the potential performance impact of each

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

28

This lesson covered the various types of joins that can be performed on tables, how to minimize data movement during joins, the performance impact of various joins depending on data and table definition, and commonly seen join methods used in query plans once the optimizer has defined the best plan for your query.

## Module 8: Performance Analysis and Tuning

### Lesson 2: Database Tuning

In this lesson, you examine best practices, commands, and tools that can be used to help tune the Greenplum Database system.

Upon completion of this lesson, you should be able to:

- Identify key steps in approaching performance tuning
- List common factors that can affect performance
- Identify best practices, commands, and tools to help tune the Greenplum Database system

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

29

From time to time, you may experience unsatisfactory performance in your environment. You should be aware of the factors that affect performance as well as how to address them.

In this lesson, you:

- Identify steps in approaching performance tuning.
- Identify common factors that can affect performance in the Greenplum Database system.
- Identify best practices, commands, and tools to help you tune your environment.

## Approaching a Performance Tuning Initiative

The following key points should be followed when tuning:

- Set performance expectations by defining goals
- Set benchmarks
- Know your baseline hardware performance for throughput and capacity
- Know your workload:
  - Heavy usage times
  - Resource contention
  - Data contention
- Focus your optimizations

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

30

Performance is the rate at which the database management system (DBMS) supplies information to those requesting it.

When approaching performance and tuning:

- **Set expectations** – Without setting an acceptable threshold for database performance, you will be in a position where you define unattainable goals. Determine what is considered *good performance* in your particular environment. This could include setting an expected query response time or defining throughput goals.
- **Benchmark** – To maintain good performance or improve performance issues, you must know the capabilities of your DBMS on a defined workload. A benchmark is a predefined workload that produces a known result set, which can then be used for comparison purposes. Periodically running the same benchmark tests can help identify system-related performance degradation over time. Benchmarks can also be used as a comparison to other workloads in an effort to identify queries or applications in need of optimization.

## Approaching a Performance Tuning Initiative (Continued)

- **Hardware** – Database performance relies heavily on disk I/O and memory usage. Knowing the baseline performance of the hardware on which your DBMS is deployed is essential in setting performance expectations. Performance of hardware components such as CPUs, hard disks, disk controllers, RAM, and network interfaces, as well as the interaction of these resources, can have a profound effect on how fast your database performs. The majority of database performance problems are caused not by the database itself, but by the underlying systems on which the database is running. I/O bottlenecks, memory problems, and network problems can significantly degrade database performance. Therefore, it is important to know the baseline capabilities of your hardware and operating system (OS). This will help you to identify and troubleshoot hardware-related problems before undertaking database-level or query-level tuning initiatives. A system's throughput defines its overall capability to process data. DBMS throughput can be measured in queries per second, transactions per second, or average response times. DBMS throughput is closely related to the processing capacity of the underlying systems (disk I/O, CPU speed, memory bandwidth, and so on), so it is important to know the throughput capacity of your hardware when setting DBMS throughput goals.
- **Workload** – Your workload equals the total demand from the DBMS. The total workload is a combination of ad-hoc user queries, applications, batch jobs, transactions, and system commands directed through the DBMS at any given time. Workload, or demand, can change over time. For example, it may increase when month-end reports need to be run, or decrease on weekends when most users are out of the office. Workload is a major influence on database performance. Knowing your workload and peak demand times will help you plan for the most efficient use of your system resources, and enable the largest possible workload to be processed.  
Contention is the condition in which two or more components of the workload are attempting to use the system in a conflicting way. For example, trying to update the same piece of data at the same time, or running multiple large workloads at once that compete with each other for system resources causes contention. As contention increases, throughput decreases.
- **Optimization** – Once you have determined what your system is capable of and what kind of performance you expect, you can focus your tuning initiatives on those areas and queries that fall short of your performance goals. Things like SQL formulation, database parameters, table design, data distribution, and so on should be altered only when performance is not acceptable.

## Common Causes of Performance Issues

The following are common causes of performance issues:



Hardware issues / failed segments



Resource allocation



Contention between concurrent workloads



Inaccurate database statistics



Uneven data distribution



SQL formulation



Database design

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

32

The following are common performance issues:

- **Hardware Problems** – As with any database system, the performance of Greenplum Database is dependant upon the hardware and IT infrastructure on which it is running. Performance will be as fast as the slowest host in the Greenplum array. Problems with CPU utilization, memory management, I/O processing, or network load will affect performance. Many hardware failures will cause the segment instances running on that hardware to fail. If mirrors are enabled, this can mean other segment hosts are doing double-duty.
- **Resource Allocation** - A database system has a limited capacity of CPU, memory, and disk I/O resources. When multiple workloads compete for access to these resources, database performance suffers. Resource allocation, also referred to as workload management, can be used to maximize system throughput while still meeting varied business requirements. Greenplum Database workload management limits the number of active queries in the system at any given time in order to avoid exhausting system resources. This is accomplished by creating role-based resource queues. A resource queue has attributes that limit the size and/or total number of queries that can be executed by the users (or roles) in that queue. By assigning all of your database roles to the appropriate resource queue, administrators can control concurrent user queries and prevent the system from being overloaded.

## Common Causes of Performance Issues

The following are common causes of performance issues:



Hardware issues / failed segments



Resource allocation



Contention between concurrent workloads



Inaccurate database statistics



Uneven data distribution



SQL formulation



Database design

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

33

- **Contention Between Concurrent Workloads** - Contention arises when two or more users or workloads try to use the system in a conflicting way. For example, if two transactions are trying to update the same table at once. A transaction seeking either a table-level or row-level lock will wait indefinitely for conflicting locks to be released. This means it is a bad idea for applications to hold transactions open for long periods of time (e.g., while waiting for user input). Are indexes being used correctly?
- **Inaccurate Database Statistics** - Greenplum Database uses a cost-based query planner that relies on database statistics. Accurate statistics allow the query planner to better estimate the number of rows retrieved by a query in order to choose the most efficient query plan. Without database statistics, the query planner can not estimate how many records might be returned, and therefore cannot assume it has sufficient memory to perform certain operations such as aggregations. In this case, the planner always takes the safe route and does aggregations by reading/writing from disk, which is significantly slower than doing them in memory. The ANALYZE command collects statistics about the database needed by the query planner.
- **Uneven Data Distribution** - When you create a table in Greenplum Database, it is important to declare a distribution key that allows for even data distribution across all segments in the system. Because the segments work on a query in parallel, Greenplum Database will always be as fast as the slowest segment. If the data is unbalanced, the segments that have more data will return their results slower.

## Common Causes of Performance Issues

The following are common causes of performance issues:



Hardware issues / failed segments



Resource allocation



Contention between concurrent workloads



Inaccurate database statistics



Uneven data distribution



SQL formulation



Database design

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

34

- **Poor SQL Formulation** - Many application developers simply write SQL statements off the top of their heads without giving much thought to designing them for efficient processing. Statements that are not designed can introduce enormous performance issues. For example, are you retrieving more data than you actually need to produce the desired result set? Are you using expensive operations that can be written in a more efficient way?
- **Poor Database Design** - Many performance issues are caused by poor database design. Examine your database design and ask yourself the following: Does the schema reflect the way the data is accessed? Can larger tables be broken down into smaller tables or partitions? Are you using the smallest data type possible to store column values? Are columns used to join tables of the same data type? Are indexes being used correctly?

## Hardware Issues

Common hardware failures include:

- Disk failures
- Host failures
- Network failures
- OS not tuned for Greenplum
- Disk Capacity:
  - 70% maximum recommended
  - VACUUM after updates, deletes and loads
- VACUUM configuration parameters

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

35

Hardware failures can cause segments to fail or degrade in performance, including:

- **Disk failure** – Although a single disk failure should not dramatically effect database performance if you are using RAID – there is some impact caused by disk resynching consuming resources on the host with failed disks.
- **Host failure** – When a host is offline (power failure, etc) the segments on that host are out of operation. This means that other hosts in the array are doing double duty as they are running both the primary segments and a number of mirrors. If mirrors are not enabled – service is interrupted. There is a temporary interruption of service to recover failed segments.
- **Network failure** – Failure of a NIC, switch, or DNS server can bring down segments.
- Capacity issues – Disk capacity on segment hosts should be at the maximum 70% capacity for optimal performance. To reclaim space after load or updates run VACUUM on a regular basis. Monitor capacity to make sure you do not run out of disk space on a segment host.
- **Not using VACUUM** – Because of the MVCC transaction concurrency model, data rows that are deleted or updated still occupy physical space on disk even though they are not visible to any new transactions. If you have a database with lots of updates and deletes, you will generate a lot of dead rows. Dead rows are held in what is called the free space map. The free space map must be sized large enough to cover the dead rows of all tables in your database. If not sized large enough, space occupied by dead rows that overflow the free space map cannot be reclaimed by a regular VACUUM command.

## Hardware Issues – VACUUM Configuration Parameters

Set the following VACUUM configuration parameters:

- max\_fsm\_relations:
  - This parameter should be set to *tables + indexes + system tables*
  - Sets the number of relations for which free space will be tracked in the memory free-space map
- max\_fsm\_pages:
  - This parameter is equal to  $16 * \text{max\_fsm\_relations}$
  - Sets the number of disk pages for which free space will be tracked

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

36

The parameters, max\_fsm\_pages together with max\_fsm\_relations control the size of the shared free space map, which tracks the locations of unused space in Greenplum Database. An undersized free space map may cause Greenplum Database to consume increasing amounts of disk space over time, because free space that is not in the map cannot be reused; instead Greenplum Database will request more disk space from the operating system when it needs to store new data.

Note that for the autovacuum daemon, VACUUM cannot run in a transaction. It creates and commits its own transactions as it goes so it can release locks as soon as possible. This is a bit tricky in Greenplum Database in keeping the segment databases in sync with each other. Because VACUUM runs asynchronously between the segment instances, it is not possible to protect tables on the segment by taking a table-level lock on the master. This is an issue with the autovacuum daemon as it acts locally on the segment without taking into account the state of the entire array.

## Resource Allocation and Contention

To work around resource allocation issues:

- Greenplum resource queues
  - Limit active queries in the system
  - Limit the size of a query a particular user can run
- Perform admin tasks at low usage times
  - Data loading, ETL
  - VACUUM and ANALYZE
  - Backups
- Design applications to prevent lock conflicts
  - Concurrent sessions not updating the same data at the same time
- Set resource-related configuration parameters

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

37

To limit resource contention for both hardware and database resources requires some knowledge of your workload, users and peak usage times. To reduce contention you can:

- Create resource queues that limit the number of active queries and/or size of queries let into the system. Then you can assign all of your database roles to a queue.
- Do not compete with database users by performing admin tasks at peak usage times. Do admin operations and load after hours.
- Do not run workloads that try to update the same resources at the same time. Lock conflicts will minimize throughput for competing workloads.

## Setting Resource Related Configuration Parameters

Resource-related configuration parameters include:

- `work_mem = 32MB`
- `maintenance_work_mem = 64MB`
- `shared_buffers = 125MB`

### Example: Set and reset a configuration parameter

```
=# SET work_mem TO '200MB';
=# ...SQL statements...
=# RESET work_mem;
```

### Example: Set a configuration parameter for a role

```
ALTER ROLE admin SET maintenance_work_mem = 100000;
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

38

The following resource-related configuration parameters can be used for resource allocation and contention:

- The `work_mem` parameter specifies the amount of memory to be used by internal sort operations and hash tables before switching to temporary disk files. Note that for a complex query, several sort or hash operations might be running in parallel; each one will be allowed to use as much memory as this value specifies before it starts to put data into temporary files. Several running sessions could be doing such operations concurrently. Therefore, the total memory used could be many times the value of `work_mem`. It is necessary to keep this fact in mind when choosing the value.

Sort operations are used for `ORDER BY`, `DISTINCT`, and merge joins. Hash tables are used in hash joins, hash-based aggregation, and hash-based processing of `IN` subqueries. Note that `work_mem` is only held for the duration of an operation, and then is immediately freed. A recommended approach is to increase `work_mem` on a per-session or per-query basis as needed.

- `maintenance_work_mem` – Specifies the maximum amount of memory to be used in maintenance operations, such as `VACUUM` or `CREATE INDEX`. Since only one of these operations can be executed at a time by a database session, and an installation normally does not have very many of them happening concurrently, it is safe to set this value significantly larger than `work_mem`. Larger settings may improve performance for vacuuming and for restoring database dumps.

## **Setting Resource Related Configuration Parameters (Continued)**

- `shared_buffers` – Sets the amount of memory a Greenplum server instance uses for shared memory buffers. This setting must be at least 128 kilobytes and at least 16 kilobytes times `max_connections`. Shared buffers define a block of memory that Greenplum Database will use to hold requests that are waiting for kernel buffer and CPU. Greenplum Database relies mostly on the OS to cache data files, therefore this parameter is set relatively low when compared to the total RAM available on a host.

## Setting Memory Management Parameters

Memory management parameters include:

- `statement_mem = 125 MB`
- `max_statement_mem = 2000 MB`  
`(segment_physical_memory/`  
`average_number_concurrent_queries)`
- `gp_vmem_protect_limit = 8192`  
`(X * physical_memory )/primary_segments`



**Note:** These parameters are used by Greenplum only when `gp_resqueue_memory_policy` is set to `eager_free` or `auto`.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

40

The new workload management system, enabled by setting the configuration parameter, `gp_resqueue_memory_policy` to `eager_free` or `auto`, ignores the `work_mem` and `maintenance_work_mem`. The resource queues allow you to specify limitations on memory and CPU resources. However, you can set configuration parameters to ensure that queries have enough resources, despite the settings on the resource queues. Configuration parameters that can be used to affect resources outside of the resource queues include:

- **statement\_mem** – The default memory allotment defined by the resource queue can be overridden on a per query basis by setting the `statement_mem` parameter. The query can be allocated memory up to the amount defined in the `max_statement_mem` parameter. These values should never exceed the physical memory of a segment host. Setting this parameter for a specific query prevents out of memory errors that could occur on a segment for that specific query.
- **max\_statement\_mem** – The parameter defines the maximum amount of memory that can be allocated to a query. This value is used both by the `statement_mem` configuration parameter and the resource queue memory limits as a ceiling for the amount of memory that can be allocated. This parameter can also be used to avoid out of memory errors if the `statement_mem` or memory limits in the resource queue are set too high.
- **gp\_vmem\_protect\_limit** – This parameter sets the maximum amount of memory that all postgres processes of an active segment instance can consume.

## Database Statistics – ANALYZE

Greenplum:

- Uses a statistics-based query planner
- Collects information such rows and range of values
- Uses `ANALYZE` to collect statistics. It should be run after:
  - Data loads
  - Restores from backups
  - Changes to schema
  - Inserts, updates, or deletes

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

41

Greenplum uses a cost-based query planner that relies on database statistics. Statistics such as number of rows and the range of values in a column are needed by the query planner for it to choose the most optimal plans.

The `ANALYZE` command collects statistics about the database needed by the query planner. It is a good idea run `ANALYZE` after any changes to the database are made, including:

- Performing data loads
- Restore data from backups
- Making changes to the schema
- Changing data with insertions, updates, or deletions

## Configuring Statistics Collection

Use the following to configure statistics collection:

- `default_statistics_target = 25`
- `gp_analyze_relative_error = .25`
- **On specific table columns, run:**

```
ALTER TABLE name ALTER column SET STATISTICS #;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

42

The following can be used to configure statistics collection for the query planner:

- **The `default_statistics_target` parameter** – Sets the default statistics target for table columns that have not had a column-specific target set using `ALTER TABLE SET STATISTICS`. Larger values increase the time needed to perform `ANALYZE`, but may improve the quality of the query planner's estimates. Specifically, this sets the maximum number of most common values and histogram bounds, or the number of value groups of approximately equal population, that are sampled. Most queries retrieve only a fraction of the rows in a table, due to having `WHERE` clauses that restrict the rows to be examined. The planner makes an estimate of the selectivity of `WHERE` clauses, or the fraction of rows that match each condition in the `WHERE` clause. A higher value, especially for columns with irregular data patterns, may allow for more accurate query plans.
- **The `gp_analyze_relative_error` parameter** – Sets the estimated acceptable error in the cardinality of the table. A value of 0.5 is equivalent to an acceptable error of 50%, the default value used in PostgreSQL. If the statistics collected during `ANALYZE` are not producing good estimates of cardinality for a particular table attribute, decreasing the relative error fraction, or accepting less error, tells the system to sample more rows to determine the number of distinct non-null data values in a column (as stored in the `n_distinct` column of the `pg_stats` table).
- **`ALTER TABLE SET STATISTICS`** – This is similar to setting `default_statistics_target` higher for a particular table column. This is recommended for columns frequently used in query predicates and joins.

## Greenplum Data Distribution

When working with data:

- Consider your table distribution key
- Check for data skew and avoid, if possible, unbalanced data
- Rebalancing a table if necessary

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

43

There are several key points to consider when working with your data:

- Consider what your table distribution key should be. In a Greenplum Database table, one or more columns are used as the distribution key, meaning those columns are used by the hashing algorithm to divide the data amongst all of the segments. The benefit of data distribution is to get your data as flat as possible.
- Check for data skew. Your data may become unbalanced, causing one segment to work harder than another. This ties into the table distribution key selection.
- If necessary, rebalance the table to achieve a more balanced solution.

## Greenplum Data Distribution – Consider the Table Distribution Key

When deciding on the table distribution key, look for:

- Even data distribution, where:
  - All segments should contain equal portions of data
  - The distribution key is unique for each record
- Local over distributed operations, where:
  - It is faster if the work can be performed at the segment level
  - A common distribution key improves joining or sorting
  - Local operations can be 5 times faster than distributed operations
- Even query processing, where:
  - All segments handle an equal amount of the query workload
  - Distribution policy and query predicates are well matched

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

44

The following considerations should be taken into account when declaring a distribution key for a table (listed in order of importance):

- **Even data distribution** — For the best possible performance, all of the segments should contain equal portions of data. If the data is unbalanced or skewed, then the segments with more data will have to work harder to perform their portion of the query processing. To ensure an even distribution of data, you want to choose a distribution key that is unique for each record, such as the primary key.
- **Local and distributed operations** — During query processing, it is faster if the work associated with join and aggregation operations can be done locally at the segment-level rather than at the system-level (distributing tuples amongst the segments). When tables share a common distribution key in Greenplum Database, joining or sorting on their shared distribution key columns will result in the most efficient query processing, as the majority of the work is done locally at the segment-level. Local operations are approximately 5 times faster than distributed operations.

## **Greenplum Data Distribution – Consider the Table Distribution Key (Continued)**

- **Even query processing** — When a query is being processed, you want all of the segments to handle an equal amount of the query workload to get the best possible performance. In some cases, query processing workload can be skewed if the table's data distribution policy and the query predicates are not well matched. For example, suppose you have a table of sales transactions. The table is distributed based on a column that contains corporate names as values. The hashing algorithm distributes the data based on the values of the distribution key, so if a predicate in a query references a single value from the distribution key, the work in the query will run on only one segment. This may be a viable distribution policy if your query predicates tend to select data on a criteria other than corporation name. However, for queries that do use corporation name in their predicates, you can potentially have just one segment instance handling all of the query workload.

## Greenplum Data Distribution – Check for Data Skew

Check for data skew using:

- `gp_toolkit.gp_skew_coefficients`
- `gp_toolkit.gp_skew_idle_fractions`
- System tools using `gpssh` to run them on multiple systems:
  - `top`
  - `iostat`

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

46

One indication of uneven table distribution is when one segment is taking significantly longer to complete its portion of a query. You can view individual segment host performance during query processing by accessing:

- **`gp_toolkit.gp_skew_coefficients`** – This view shows data distribution skew by calculating the coefficient of variation (CV) for the data stored on each segment. This view is accessible to all users, however non-superusers will only be able to see tables that they have permission to access.  
The column, skccoeff, shows the coefficient of variation (CV). This is calculated as the standard deviation divided by the average. It takes into account both the average and variability around the average of a data series. The lower the value, the better. Higher values indicate greater data skew.
- **`gp_toolkit.gp_skew_idle_fractions`** – This view shows data distribution skew by calculating the percentage of the system that is idle during a table scan, which is an indicator of processing data skew. This view is accessible to all users, however non-superusers will only be able to see tables that they have permission to access.  
The column, siffracton, shows the percentage of the system that is idle during a table scan. This is an indicator of uneven data distribution or query processing skew. For example, a value of 0.1 indicates 10% skew, a value of 0.5 indicates 50% skew, and so on. Tables that have more than 10% skew should have their distribution policies evaluated.
- System tools such as `top` and `iostat` can be run on multiple systems with `gpssh`.

The `gp_toolkit` schema is an administrative schema that can be used to query system catalogs, log files, and the operating system for system status information.

## Greenplum Data Distribution – Rebalancing a Table

Rebalancing a table can be performed with the following:

- Change the distribution policy to a different column and redistribute the table and child tables:

```
ALTER TABLE sales SET DISTRIBUTED BY (customer_id);
```

- Redistribute table data to correct data skew:

```
ALTER TABLE sales SET WITH (REORGANIZE=TRUE);
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

47

You can rebalance a table to change the distribution policy that previously existed on the table. If you need to change the distribution policy, use the `ALTER TABLE` SQL command and the `SET DISTRIBUTED BY` clause. This automatically redistributes the data.

To redistribute the data for tables with a random policy or to simply rebalance the table, use `REORGANIZE=TRUE` as part of the `ALTER TABLE` SQL command.

## SQL Formulation – General Considerations

When creating your queries:

- Know your data
- Minimize returned rows
- Avoid unnecessary columns in the result set
- Avoid unnecessary tables
- Avoid sorts of large result sets
- Match data types in predicates

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

48

When creating queries:

- **Know your data** - When writing the SQL statement, you should have some idea of how many rows are in each table, how selective your WHERE predicates are, and how many rows you expect in the result set. The larger the number of rows involved, the more time you should spend thinking about the best way to write the SQL statement.
- **Minimize returned rows** – The fewer the rows in the result set, the more efficiently the query will run. Verify your query only fetches the data you need. If returning a large number of rows is unavoidable, consider using cursors to fetch a subset of rows at a time. Avoid multiple DISTINCT aggregates if possible.
- **Avoid unnecessary columns in the result set** – The wider the data in each row in the result set, the more disk space and memory that is required for intermediate operations such as sorts to hold the result set.
- **Avoid unnecessary tables** – The fewer the tables, the more efficient the query.
- **Avoid sorts of large result sets if possible** – Sorts are expensive, especially when the rows being sorted will not fit in memory. Try to limit the amount of data that needs to be sorted.
- **Match data types in predicates** – When a query predicate compares two column values as is done with joins, it is important for the data types to match. When the data types do not match, the DBMS must convert one of them before performing the comparison, and while the work to do this is relatively small, it can add up when it has to be done for a large number of rows.

## SQL Formulation – Greenplum Specific Considerations

Greenplum-specific guidelines for creating queries include:

- Use common distribution keys:
  - For joins and aggregations
  - So most of the work is performed at the segment level
- Consider the table data distribution policy and query predicates:
  - To have segments handle an equal amount of work
  - To provide the best possible performance

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

49

Greenplum-specific considerations for creating queries include:

- Using common distribution key columns for joins and aggregations when possible. When tables share a common distribution key in Greenplum Database, joining or sorting on their shared distribution key columns will result in the most efficient query processing. This allows the majority of the work to be done at the segment-level, which is significantly faster than redistributing data across the segments at the system-level.
- Considering the table data distribution policy when determining query predicates. When a query is being processed, you want all of the segments to handle an equal amount of the query workload to get the best possible performance. In some cases, query processing workload can be skewed if the table's data distribution policy and the query predicates are not well matched. For example, suppose you have a table of sales transactions. The table is distributed based on a column that contains corporate names as values. The hashing algorithm distributes the data based on the values of the distribution key, so if a predicate in a query references a single value from the distribution key, the work in the query will run on only one segment. This may be a viable distribution policy if your query predicates tend to select data on a criteria other than corporation name. However, for queries that do use corporation name in their predicates, you can potentially have just one segment instance handling all of the query workload.

## Database Design

When considering the database design:

- Select appropriate data types
- Use a denormalized model
- Consider table partitioning
- Reconsider the use of indexes

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

50

When designing the database, consider the following:

- **Data Types** – The data type of a column determines the types of data values that column can contain.
- **Denormalization** – In relational database theory, normalization is the process of restructuring the logical data model of a database to eliminate redundancy and improve data organization. A denormalized scheme, while introducing redundancy, can improve performance.
- **Table partitioning** – This addresses the problem of large fact tables by dividing the table into more manageable pieces. Partition tables can have a positive effect on query performance.
- **Indexing** – In distributed databases such as Greenplum Database, indexes should be used sparingly, as they do not offer the same level of performance that may be seen in other traditional databases.

## Database Design – Selecting Appropriate Data Types

When selecting data types, choose a data type:

- That uses the least possible space
- That best constrains the data:
  - Use character data types for strings
  - Use date or timestamp data types for dates
  - Use numeric data types for numbers
  - Use TEXT or VARCHAR for character data
- Use identical data types for columns used in cross-table joins

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

51

The data type of a column determines the types of data values that column can contain. As a general rule, you should choose the data type that:

- Uses the least possible space, yet can still accommodate your data
- That best constrains the data in that column.

For example, use:

- Character data types for strings
- Date or timestamp data types for dates
- Numeric data types for numbers. Using test data types for numbers introduces more overhead than needed – character set and locale checks. For numeric column data types, use the smallest data type in which the data will fit. A lot of space is wasted if, for example, the BIGINT data type is used when the data would always fit in INT or SMALLINT.
- For character column data types, there are no performance differences between the use of CHAR, VARCHAR, and TEXT data types, apart from the increased storage size when using the blank-padded type. While CHAR has performance advantages in some other database systems, it has no such advantages in Greenplum Database. In most situations, TEXT or VARCHAR should be used instead.

## **Database Design – Selecting Appropriate Data Types (Continued)**

- Use identical data types for the columns you plan to use in cross-table joins. Joins work much more efficiently if the data types of the columns used in the join predicate (usually the primary key in one table and a foreign key in the other table) have identical data types. When the data types are different, the database has to convert one of them so that the data values can be compared correctly, and such conversion amounts to unnecessary overhead.

Refer to the *Greenplum Database Administrator Guide* for a list of the built-in data types available in Greenplum Database and their associated sizing information.

## Database Design – Denormalization

Normalization:

- Is the process of eliminating redundancy and improving data organization
- Is used by online transaction processing (OLTP) databases

Denormalization:

- Is used by online analytical processing (OLAP) databases
- Translates into redundant data
- May facilitate ease of use and performance
- Is used by the star schema, where:
  - Data is stored in a central fact table
  - Dimension tables are denormalized
  - Complexity of queries is reduced
  - ETL processing may be required

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

53

In relational database theory, normalization is the process of restructuring the logical data model of a database to eliminate redundancy and improve data organization. The goals of normalization are to improve data integrity, eliminate repeating data, and reduce the potential for anomalies during database operations.

Databases intended for Online Transaction Processing (OLTP) are typically more normalized than databases intended for Online Analytical Processing (OLAP). OLTP and OLAP systems have very different requirements. OLTP applications are characterized by a high volume of small transactions, each transaction leaving the database in a consistent state. By contrast, databases intended for OLAP operations are primarily read-only databases. OLAP applications tend to extract historical data that has accumulated over a long period of time. For OLAP databases, redundant or denormalized data may facilitate ease-of-use and performance.

Data warehouses often use denormalized or partially denormalized schemas (such as a star schema) to optimize query performance. In a star schema design, data is stored in a central fact table surrounded by one or more denormalized dimension tables. One advantage of a star schema design is improved performance because the data is pre-joined, reducing the complexity of the queries. Also, because of the simplicity of the design, it is easy for business users to understand and use. One disadvantage of the dimensional approach is that some type of ETL process is required to guarantee data consistency.

## Database Design – Table Partitioning

Table partitioning:

- Addresses the problem of supporting very large tables
- Divides large tables into smaller, manageable pieces
- Can improve query performance
- Lets the query planner scan only relevant data
- Should be used to help selectively scan data based on query predicates

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

54

Table partitioning addresses the problem of supporting very large tables, such as fact tables, by allowing you to divide them into smaller and more manageable pieces.

Partitioned tables can improve query performance by allowing the Greenplum Database query planner to scan only the relevant data needed to satisfy a given query rather than scanning the entire contents of a large table. Partitioned tables can also be used to facilitate database maintenance tasks, such as rolling old data out of the data warehouse. Keep in mind that partitioning should be used to help selectively scan data based on query predicates. Any query that does not scan according to the partition design will not benefit from this approach and can even slightly lessen performance. In this case, there are more scans performed but the scan results are not used to filter data partitions.

## Database Design – Indexes

If you are considering indexes, use the following guidelines:

- Use sparingly in Greenplum Database
- Test the query workload without indexes
- Ensure any indexes added are used by the query workload
- Verify that indexes improve query performance
- Indexes can improve performance of OLTP type workloads

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

55

Greenplum Database is very fast at sequential scanning, where as indexes use a random seek pattern to locate records on disk. As the data is distributed across segments, each segment scans a smaller portion of the overall data in order to get the desired result. If you are using table partitioning, the total data to scan may be even a fraction of that.

Follow these guidelines when considering indexes:

- Test your query workload without adding any additional indexes. Greenplum Database will automatically create PRIMARY KEY indexes for tables with primary keys. If you experiencing unsatisfactory performance, then you may try adding indexes to see if performance improves. Indexes do add some database overhead. Indexes:
  - Consume storage space.
  - Must be maintained whenever the table is updated.
- Ensure indexes you create are being used by your query workload.
- Verify that the indexes you add improve query performance, compared to a sequential scan of the table. You can do this by executing EXPLAIN on a query to see its plan.

EXPLAIN will be covered later in the course.

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 query workload returns very large data sets. It therefore does not make efficient use of indexes. For this type of workload, it is better to use sequential scans to locate large chunks of data on disk rather than to randomly seek the disk using index scans.

## Database Design – Index Considerations

When incorporating indexes, use the following guidelines:

- Avoid using indexes on frequently updated columns
- Avoid overlapping indexes
- Use bitmap indexes where applicable instead of B-tree
- Drop indexes for loads
- Consider a clustered index
- Configuring index usage with the following:  
`enable_indexscan = on | off`
- Compressed append-optimized tables may benefit from indexes
- If indexing partitioned tables, index columns should not be the same as partition columns

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

56

Consider the following when incorporating indexes into your design:

- Avoid indexes on frequently updated columns. Creating an index on a column that is frequently updated increases the amount of writes required when the column is updated.
- Use Bitmap indexes for low selectivity columns. Greenplum Database has an additional index type called a Bitmap index, which is a fraction of the size of B-tree indexes for low cardinality columns. Performance is boosted for queries that use multiple predicates to filter results of bitmap indexed columns. Filtering is done before row is fetched.
- Avoid overlapping indexes. Overlapping indexes (those that have the same leading column) are redundant and unnecessary.
- Drop indexes for loads. For mass loads of data into a table, consider dropping the indexes and re-creating them after the load is complete. This is often faster than updating the indexes.

## Database Design – Index Considerations (Continued)

- **Clustered index.** Clustering an index means that the records are physically ordered on disk according to the index. If the records you need are distributed randomly on disk, then the database has to seek across the disk to get the records requested. If those records are stored more closely together, then the fetching from disk is more sequential. A good example for a clustered index is on a date column where the data is ordered sequentially by date. A query against a specific date range will result in an ordered fetch from the disk, which leverages fast sequential access. Postgres has a CLUSTER command that does this, but Greenplum does not recommend its use because it takes a really long time to cluster in this way. Instead do a CREATE TABLE AS SELECT ORDER BY and order the data according to the associated column.
- **Configure index usage** – The enable\_indexscan parameter allows you to turn off use of index scan operations in query plans – useful for confirming performance gains (or losses) of using an index.
- **Indexes on append-optimized compressed tables** – Compressed data requires additional CPU cycles to uncompress when retrieving or writing data to the table. An index on a compressed append-optimized table can improve performance as only the rows that are being retrieved will be uncompressed.
- **Indexing partitioned tables** – While it is not considered a good practice to index partitioned tables, should you need to do so, the indexed column must not be the same as the partitioned column. Remember, the goal of partitioning is to reduce the number of rows that need to be scanned. The use of indexes should be to further improve the performance of finding rows in a subset of the already partitioned rows.

## Tracking Performance Issues

Performance management steps taken:

- Are often reactive
- Can focus efforts on tuning specific workloads
- Can be caused by:
  - Hardware problems
  - System failures
  - Resource contention
- Can be tracked with:
  - pg\_stat\_activity
  - pg\_locks or pg\_class
  - Database logs
  - UNIX system utilities

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

58

Many of the performance management steps taken by database administrators are reactive - a business user calls with a response time problem, a batch job fails, the system suddenly becomes unavailable. If the problem is related to a particular workload or query, then you can focus your efforts on tuning that particular workload. If the performance problem is system-wide, then hardware problems, system failures, or resource contention may be the cause.

Problems can be tracked down using a combination of the following:

- The pg\_stat\_activity view
- The pg\_locks or pg\_class views
- Database logs
- UNIX system utilities

## Tracking Performance Issues – pg\_stat\_activity System Catalog View

The pg\_stat\_activity view:

- Is a system catalog view
- Shows one row per server process

```
gpadmin@mdw:~$ datamart=# select datname, procpid, username, current_query, client_addr, application_name from pg_stat_activity where current_query !~ 'IDLE';
-[ RECORD 1 ]-----
datname      | faa
procpid     | 30695
username    | gpadmin
current_query| 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
client_addr  | 10.105.59.13
application_name| psql
-[ RECORD 2 ]-----
datname      | faa
procpid     | 927
username    | gpadmin
current_query| select count(*), gp_segment_id from factontimeperformance2 group by gp_segment_id;
client_addr  |
application_name| psql
datamart=#
```

Pivotal  
© 2015 Pivotal Software, Inc. All rights reserved.

This system view shows one row per server process . The view contains the following fields:

- Database OID
- Database name
- Process ID
- User OID
- User name
- Current query
- Time at which the current query began execution
- Time at which the process was started
- Client address
- Port number
- Application name
- Transaction start time

Querying this view can provide more information about the current workload on the system. This view should be queried as the database superuser to obtain the most information possible. Also note that the information does not update instantaneously.

The output shown displays all processes executing that are not idle. The output is formatted with the PSQL metacommand, \x.

## Tracking Performance Issues – pg\_locks

### System Activity View

The pg\_locks view:

- Is a system catalog view
- Lets you view information on outstanding locks
- Can help identify contention between sessions
- Provides a global view of all locks in the database system
- Can be joined to pg\_class.oid for relations in the current database
- Can have the pid column joined to pg\_stat\_activity.procpid for more session information

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

60

The pg\_locks system view allows you to view information about outstanding locks. Examining pg\_locks for ungranted locks can help identify contention between database client sessions. The pg\_locks view provides a global view of all locks in the database system, not only those relevant to the current database.

Although its relation column can be joined against pg\_class.oid to identify locked relations, such as tables, this will only work correctly for relations in the current database. The pid column can be joined to the pg\_stat\_activity.procpid to get more information on the session holding or waiting to hold a lock.

## Tracking Performance Issues – pg\_locks

### System Activity View (Cont)

```

gpadmin@mdw:~
faa1=# select l.pid, l.locktype, d.datname, c.relname, l.mode from pg_locks l, pg_database d, pg_class c where d.
oid=l.database and l.relation=c.oid;
 pid | locktype | datname | relname | mode
-----+-----+-----+-----+-----+
 6622 | relation | faal | pg_class_oid_index | AccessShareLock
 6622 | relation | faal | pg_class_relname_nsp_index | AccessShareLock
 6622 | relation | faal | pg_locks | AccessShareLock
 6622 | relation | faal | pg_class | AccessShareLock
 25630 | relation | faal | factontimeperformance | AccessShareLock
 25632 | relation | faal | factotperf_1_prt_y2011 | AccessShareLock
 31117 | relation | faal | factontimeperformance | AccessShareLock
 31117 | relation | faal | factotperf_1_prt_y2011 | AccessShareLock
 6616 | relation | faal | factontimeperformance | AccessShareLock
 6616 | relation | faal | factotperf_1_prt_y2011 | AccessShareLock
(12 rows)
faa1=#

```

**Result of the pg\_locks query**

**SELECT query**

**All locks in the current database are displayed**

**Result of a VACUUM**

Pivotal  
© 2015 Pivotal Software, Inc. All rights reserved.

The pg\_locks system activity view contains the following columns:

- Type of the lockable object.
- OID of the database in which the object exists.
- OID of the relation, if it exists. If it does not exist, the value for the column is NULL.
- Page number within the relation.
- Tuple number within the page.
- Transaction ID
- OID of the system catalog containing the object.
- OID of the object within the system catalog.
- Column number.
- Transaction ID that is holding or awaiting the lock.
- Process ID associated with the server process holding or awaiting the lock.
- Name of the lock.
- Boolean specifying whether the lock is held or is being waited on.
- Client session ID associated with the lock holder.
- Boolean specifying if the lock is being held by the writer process.
- Segment ID where the lock is being held.

## Tracking Performance Issues – Greenplum Database Log Files

Log files:

- Can be found for the master and segments
- Is located in the data directory location of the instance
- Can be accessed with:
  - `gpstate -l` to get the location of log files
  - `gpstate -e` to list the last lines of the log files

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

62

The log files for each Greenplum Database server instance, master and segments, can be found in the data directory location of that instance.

If you are not sure of the location of the data directories, you can use:

- `gpstate -l` to determine the location of the log files
- `gpstate -e` to view last lines of all log files

The `gpstate` command displays information about a running Greenplum Database system. It can be used to identify failed segments.

## Tracking Performance Issues – UNIX System Utilities

System monitoring tools:

- Include:
  - ps
  - top
  - iostat
  - vmstat
  - netstat
- Help to:
  - Identify processes running on the system
  - Identify the most resource intensive tasks
- Can help identify queries overloading system resources
- Can be run on several hosts at once using gpssh

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

63

System monitoring utilities such as ps, top, iostat, vmstat, and netstat can be used to monitor database activity on the hosts in your Greenplum Database array.

These tools can be used to help:

- Identify Greenplum Database processes, including postmaster/postgres processes, currently running on the system.
- The most resource intensive tasks with regards to CPU, memory, disk I/O, or network activity.

Looking at these system statistics can help identify queries that are overloading the system by consuming excessive resources and thereby degrading database performance. Greenplum Database comes with a management tool called gpssh. This allows you to run these system monitoring commands on several hosts at once.

## Module 8: Performance Analysis and Tuning

### Lesson 2: Summary

During this lesson the following topics were covered:

- Key steps in approaching performance tuning
- Common factors that can affect performance
- Best practices, commands, and tools to help tune the Greenplum Database system

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

64

This lesson discussed the key impacts and steps in approaching performance tuning and the common factors that can affect the overall performance of the system and queries. Best practices, commands, and tools, views, and tables that you can use to help tune the system and queries are also provided.

## Module 8: Performance Analysis and Tuning

### Lesson 3: Query Profiling

In this lesson, you examine the structure of a query plan.

Upon completion of this lesson, you should be able to:

- Describe query profiling
- Access and view query plans

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

65

Understanding the structure of a query plan can help you decide how best to tune the database or your queries for the fastest and most efficient response time.

In this lesson, you:

- Describe query profiling.
- Access and view query plans.

## Query Profiling

Query profiling:

- Is the act of using query plans to identify tuning opportunities
- Lets you address several key concepts

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

66

Query profiling is the act of looking at the query plan, using EXPLAIN or EXPLAIN ANALYZE, to identify tuning opportunities of poorly performing queries. You should only tackle query profiling and tuning when you are first designing your database or when you are experiencing poor performance. Query profiling requires that you know your data well. You cannot identify problems you know what to expect.

There are several major issues or key items that query profiling allows you to address.

## Query Profiling – Addressing Key Issues

Several key issues are addressed in query profiling, including:

- Plan operations that are taking exceptionally long
- Are the planner's estimates close to reality?
- Is the planner applying selective predicates early?
- Is the planner choosing the best join order?
- Is the planner selectively scanning partitioned tables?
- Is the planner choosing hash aggregate and hash join operations where applicable?
- Is there sufficient work memory?

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

67

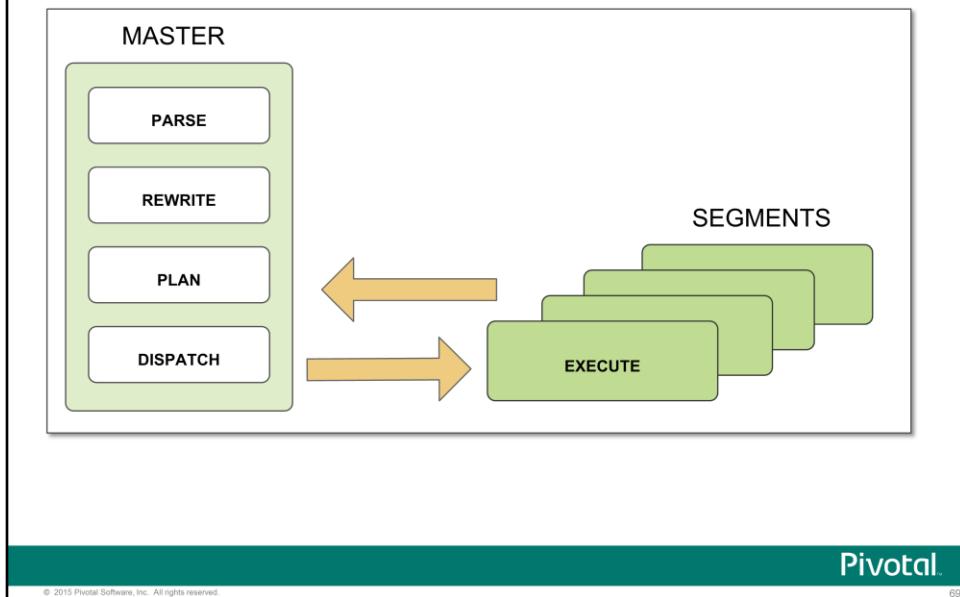
Query profiling lets you look at the following key items:

- Is there one operation in the plan that is taking exceptionally long? When looking through the query plan, is there one operation that is consuming the majority of the query processing time as compared to other operations? For example, is an index scan taking a lot longer than expected? Maybe the index needs to be REINDEXED because its out of date. Maybe see if you can force the planner to choose another operation by turning `enable_indexscan=off` temporarily and then rerunning the query.
- Are the planner's estimates close to reality? Run an `EXPLAIN ANALYZE` and see if the number of rows estimated by the planner is close to the number of rows actually returned by the query operation. If there is a huge discrepancy, you may need to collect more statistics on the relevant columns.
- Are selective predicates applied early in the plan? The most selective filters should be applied early in the plan so that less rows move up the plan tree. If the query plan is not doing a good job at estimating the selectivity of a query predicate, you may need to collect more statistics on the relevant columns. You can also try reordering the `WHERE` clause of your SQL statement.

## Query Profiling – Addressing Key Issues (Continued)

- Is the planner choosing the best join order? When you have a query that joins multiple tables, make sure that the planner is choosing the most selective join order. Joins that eliminate the largest number of rows should be done earlier in the plan so that less rows move up the plan tree. If the plan is not choosing the optimal join order you can use explicit JOIN syntax in your SQL statement to force the planner to use a specified join order. You can also collect more statistics on the relevant join columns.
- Is the planner selectively scanning partitioned tables? If you are using table partitioning, is the planner selectively scanning only the child tables required to satisfy the query predicates? Do scans of the parent tables return 0 rows (they should, since the parent tables should not contain any data).
- Is the planner choosing hash aggregate and hash join operations where applicable? Hash operations are typically much faster than other types of joins or aggregations. Row comparison and sorting is done in memory rather than reading/writing from disk.
- Is there sufficient work memory? In order for hash operations to be chosen, there has to be sufficient work memory available to hold the number of estimated rows. Try increasing work memory to see if you can get better performance for a given query. If possible run an EXPLAIN ANALYZE for the query, which will show you which plan operations spilled to disk, how much work memory they used, and how much was required to not spill to disk.

## The Query Process

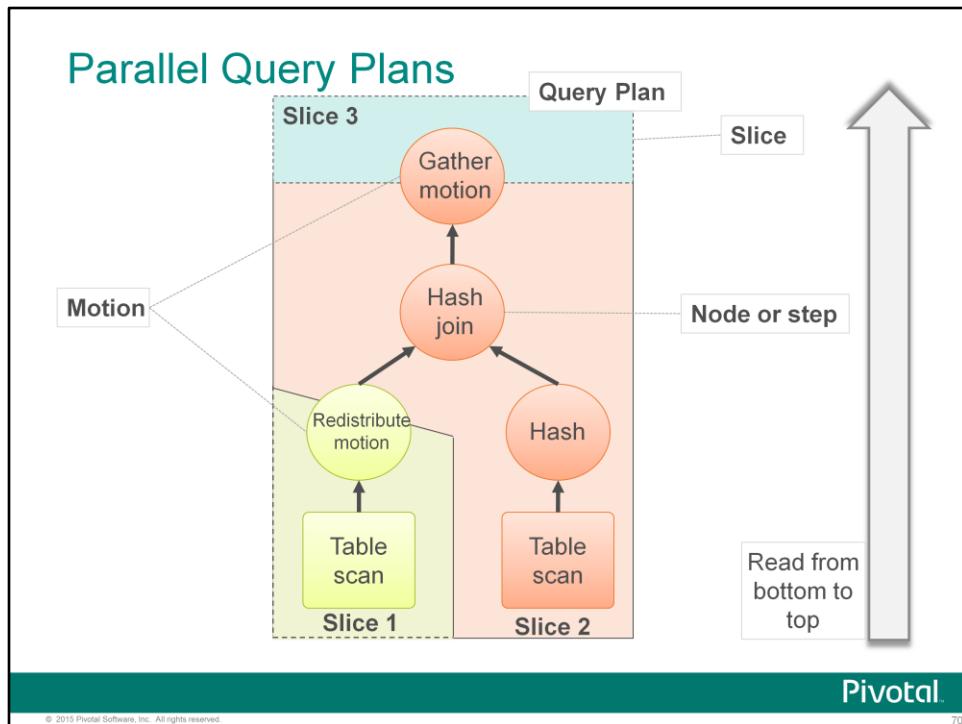


When a query is issued to Greenplum Database, the master:

- Parses the query and checks syntax.
- Sends the query to the query rewrite system where VIEWS and RULES are processed.
- Creates a parallel query execution plan.
- Slices the plan and each segment is dispatched its slice to work on.

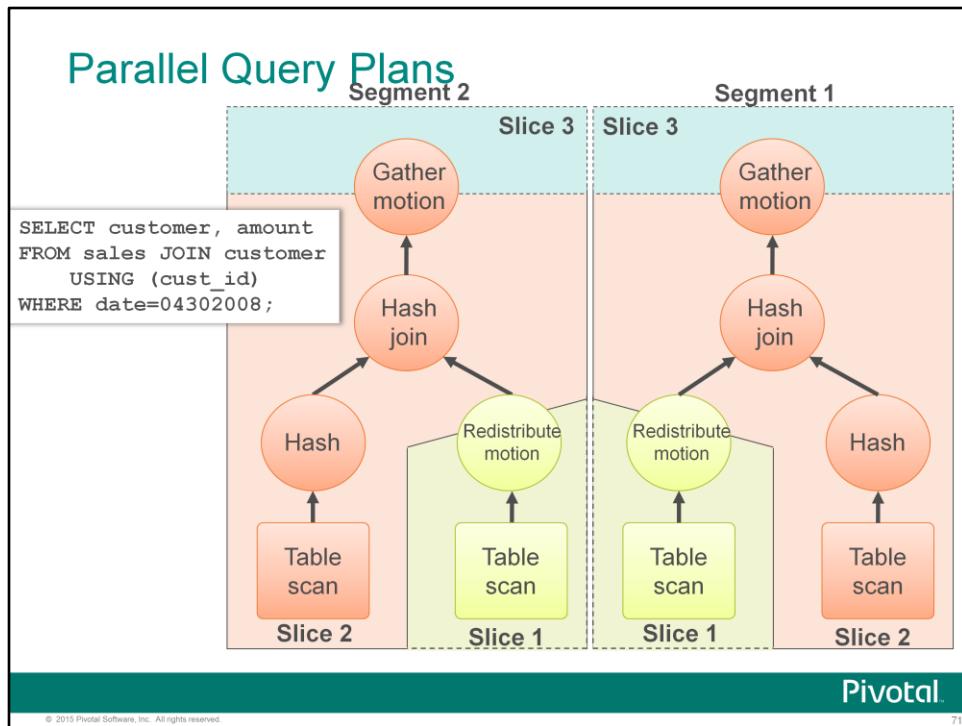
The segments execute the query between themselves, moving data around as needed over the interconnect.

When execution is complete, the segments return the results to the master. Sometimes the master may do final aggregations for small data sets only and then the master sends the result to the client. Most of the tuning of queries is focused on the planning stage of this process. You want the planner to give the best plan, and thereby the best performance possible.



A query plan:

- Is the set of operations the Greenplum Database performs to produce the answer to a given query.
- Consists of nodes or steps which represent a database operation such as a table scan, join, aggregation or sort.
- Is read and executed from bottom to top of the query plan. Typical database operations , such as tables scans, and joins include an additional operation called a *motion*.
- May include a motion operation which involves moving tuples, or rows, between the segments during query processing.
- Is divided into slices to achieve maximum parallelism during query execution. A slice is a portion of the plan that can be worked on independently at the segment level.
- Is sliced wherever there is data movement in the query plan, one slice on each side of the motion. If a join occurs on distributed tables and there is no data movement, there are no slices created. If there is data movement, slices are created for each data movement point.
- Can contain a gather motion, which is when the segments send results back up to the master for presentation to the client.



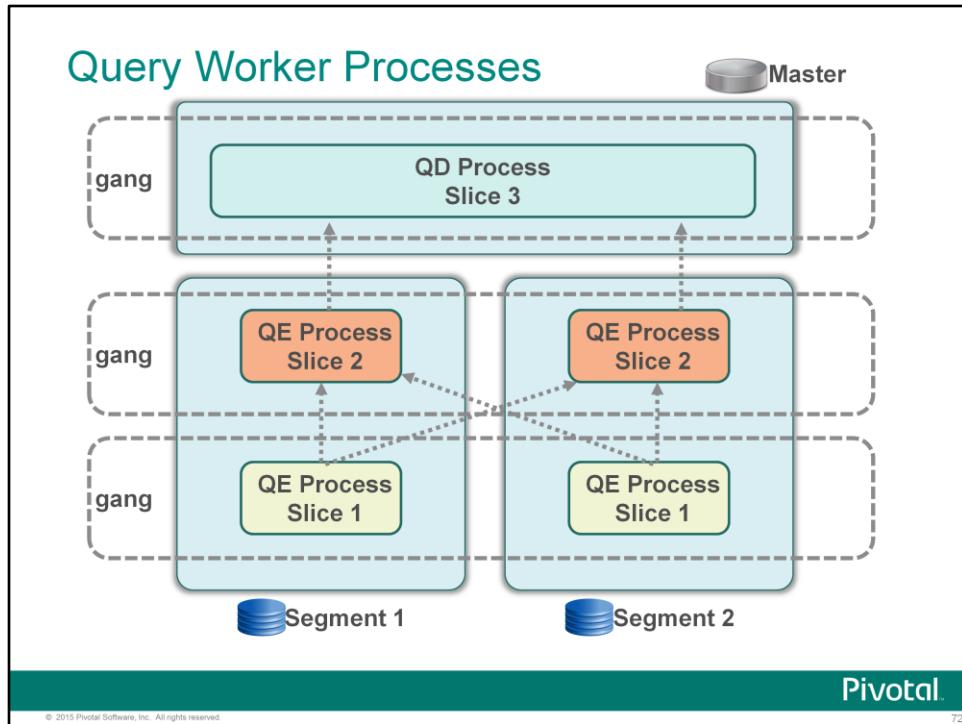
Each segment gets a copy of the query plan and works on it in parallel.

Consider the following simple query involving a join between two tables:

```
SELECT customer, amount
FROM sales JOIN customer USING (cust_id)
WHERE date=04302008;
```

There is a redistribute motion that moves tuples between the segments to complete the join. The plan is sliced on either side of the redistribute motion, creating slice 1 and slice 2. This query plan also includes a gather motion. Since a query plan is always sliced wherever a motion occurs, this plan also has an implicit slice at the very top of the plan, slice 3. Not all query plans involve a gather motion. For example, a CREATE TABLE x AS SELECT... statement would not have a gather motion (tuples are sent to the newly created table, not to the master).

**Note:** If a SORT is performed on the data, that is performed before the Gather Motion action occurs.



Greenplum creates a number of database processes, `postgres`, to handle the work of a query:

- **Query dispatcher** – On the master, the query worker process is called the query dispatcher (QD). The QD is responsible for creating and dispatching the query plan, and for accumulating and presenting the final results.
- **Query executor** – On the segments, a query worker process is called a query executor (QE). A QE is responsible for completing its portion of work and communicating its intermediate results to the other worker processes. For each slice of the query plan, there is at least one worker process assigned. A worker process works on its assigned portion of the query plan independently. During query execution, each segment will have a number of processes working on the query in parallel.

Related processes that are working on the same portion of the query plan are referred to as *gangs*. Gangs are connected by a connection identifier that can be retrieved by issuing a process listing command. Process identifiers are prefixed with the term, `con`. For example, all the gather slices in this example may show up in the process list as `con13`.

As a portion of work is completed, tuples flow up the query plan from one gang of processes to the next. This inter-process communication between the segments is what is referred to as the interconnect component of Greenplum Database.

## Viewing the Query Plan

To see the plan for a query, use:

- EXPLAIN <query>
- EXPLAIN ANALYZE <query>

Query plans:

- Are read from bottom to top
- Include motions, such as Gather, Redistribute, Broadcast on
  - Joins
  - Sorts
  - Aggregations

The following metrics are given for each operation:

- Cost (units of disk page fetches)
- Rows (rows output by this node)
- Width (byte count of the widest row produced by this node)

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

73

Use the following commands to see the query plan:

- **EXPLAIN** – The EXPLAIN command lets you view the query plan for a query.
- **EXPLAIN ANALYZE** – This command runs the query and give you the actual metrics rather than just the estimates.

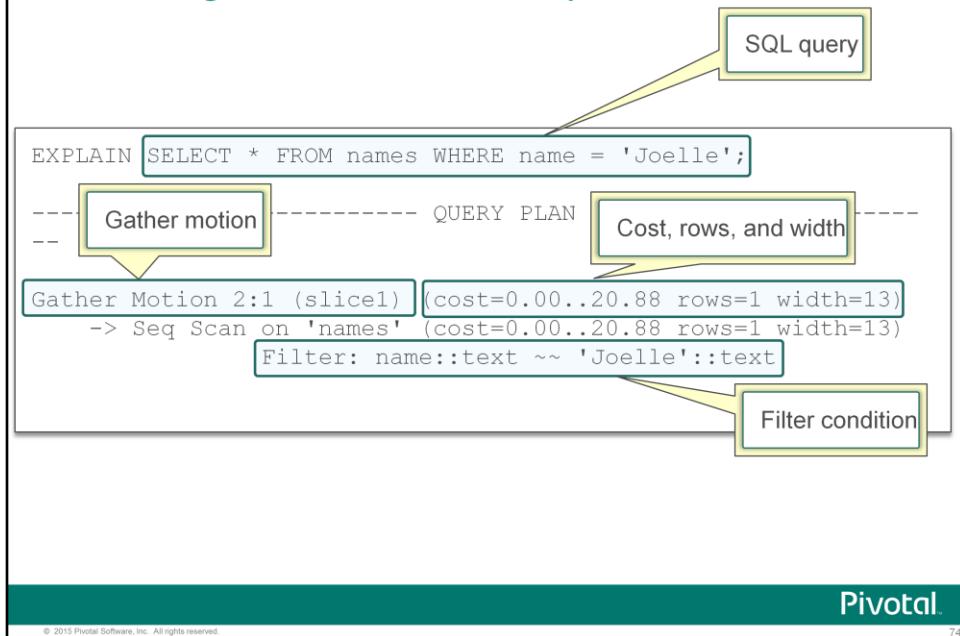
A query plan is a tree plan of nodes. Each node represents a database operation such as a table scan, join, or sort. Query plans are read from bottom to top.

Greenplum Database has an additional query plan node type called a motion node, which are the operations that move tuples between segments.

In all query plan nodes or operations, the following metrics are given:

- Cost is measured in units of disk page fetches; that is, 1.0 equals one sequential disk page read. The first estimate is the start-up cost and the second is the total cost.
- Rows is the total number of rows output by this plan node.
- Width is the width (in bytes) of the widest row produced by this plan node.

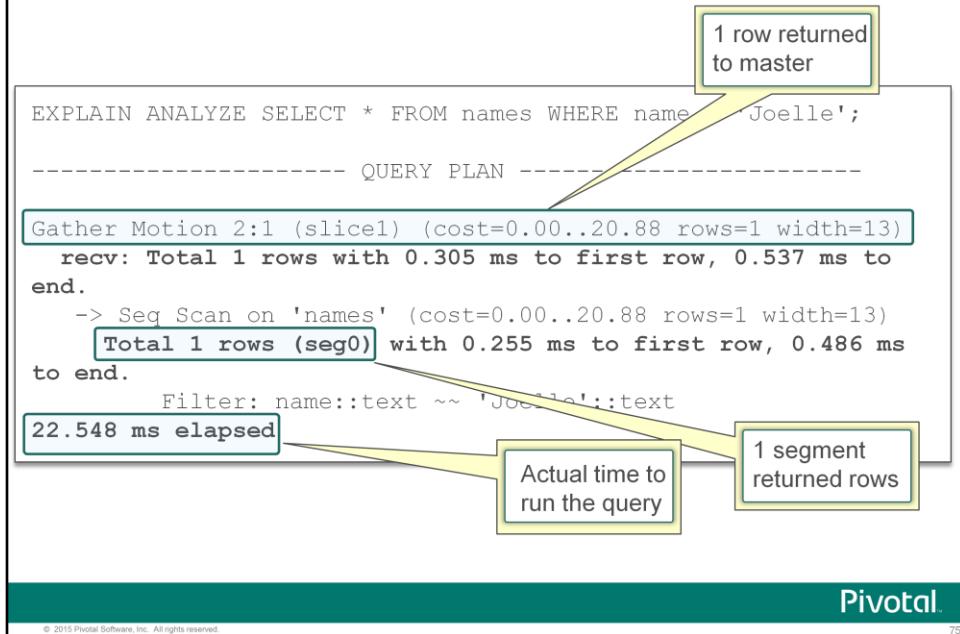
## Reading the EXPLAIN Output



To illustrate how to read an EXPLAIN query plan, consider the following example for a very simple query:

- When reading the plan from the bottom of the output, the query planner starts by doing a sequential scan of the names table.
- Note that the `WHERE` clause is being applied as a filter condition. This means that the scan operation checks the condition for each row it scans, and outputs only the ones that pass the condition.
- The results of the scan operation are passed up to a gather motion operation. In Greenplum Database, a gather motion is performed when segments send rows up to the master. In this case we have 2 segment instances sending to 1 master instance (2:1). This operation is working on slice1 of the parallel query execution plan. In Greenplum Database a query plan is divided into slices so that portions of the query plan can be worked on in parallel by the segments.
- The estimated startup cost for this plan is 00.00 (no cost) and a total cost of 20.88 disk page fetches. The planner is estimating that this query will return one row and the widest row is 13 bytes.

## Reading the EXPLAIN ANALYZE Output



To illustrate how to read an EXPLAIN ANALYZE query plan, let us examine the same simple query we used in the EXPLAIN example:

- Notice that there is some additional information in this plan that is not in a regular EXPLAIN plan. The parts of the plan in bold show the actual timing and rows returned for each plan node.
- Reading the plan from the bottom up, you will see some additional information for each plan node operation. The total elapsed time it took to run this query was 22.548 milliseconds.
- The sequential scan operation had only one segment (seg0) that returned rows, and it returned just 1 row. It took 0.255 milliseconds to find the first row and 0.486 to scan all rows. Notice that this is pretty close to the planner's estimate - the query planner estimated that it would return one row for this query, which it did.
- The gather motion operation then received 1 row (segments sending up to the master). The total elapsed time for this operation was 0.537 milliseconds.

## Lab: Database Tuning

In this lab, you use output from EXPLAIN ANALYZE to answer questions about the behavior of a query.

You will:

- Tune the database and queries

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

76

In this lab, you use output from EXPLAIN ANALYZE to answer questions about the behavior of a query.

## Module 8: Performance Analysis and Tuning

### Lesson 3: Summary

During this lesson the following topics were covered:

- Query profiling
- Accessing and viewing query plans

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

77

This lesson discussed query profiling, the major components of the query process, and the ability to access and view query plans. For every query issued to the system, the planner creates a plan based on the query, parameter settings, the environment, and the object structure. There are plenty of opportunities to tune the query to ensure that it is optimized for execution.

## Module 8: Performance Analysis and Tuning

### Lesson 4: Explain the Explain Plan – Analyzing Queries

In this lesson, you use the explain plan to determine how the optimizer will handle a query.

Upon completion of this lesson, you should be able to:

- Identify the benefits of using the Greenplum EXPLAIN and EXPLAIN ANALYZE utilities
- Decipher an explain plan based on output provided
- Analyze the query plan

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

78

In this lesson, you explore the explain plan in greater detail to see how the optimizer handles a query. By analyzing exactly what occurs with the query planner, you can see how providing the most accurate statistics can affect what the optimizer generates for the query plan.

In this lesson, you will:

- Identify the benefits of using the Greenplum EXPLAIN and EXPLAIN ANALYZE utilities.
- Decipher an explain plan based on output provided.
- Analyze the query plan.

## The Greenplum EXPLAIN and EXPLAIN ANALYZE Utilities

### The EXPLAIN utility:



- Allows the user to view how the optimizer will handle the query
- Shows the cost, in page views, and an estimated run time

### The EXPLAIN ANALYZE utility:



- Executes the query
- Shows the difference between the estimates and the actual run costs

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

79

EXPLAIN gives you a guess as to what it's going to do, based on statistics. EXPLAIN ANALYZE runs the query and tells you what it did. This does give an accurate depiction of the time taken to run the query, whereas EXPLAIN produces an estimate of the time it will take to execute the command.

## EXPLAIN or EXPLAIN ANALYZE

The following depicts when either utilities are used:

- Use EXPLAIN ANALYZE to run the query and generate query plans and planner estimates
- Run EXPLAIN ANALYZE on queries to identify opportunities to improve query performance
- EXPLAIN ANALYZE is also a engineering tool used by Greenplum developers
- Focus on the query plan elements that are relevant from an end user perspective to improve query performance

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

80

The query plan will define how the query will be executed in the Greenplum database environment. The query planner uses the database statistics to determine the query plan with the lowest possible cost. Cost is measured in disk I/O, CPU and memory. The goal is to minimize the total execution cost for the plan.

Use EXPLAIN ANALYZE to execute the query and generate the query plan with estimates. Run EXPLAIN ANALYZE on queries to identify opportunities to improve query performance.

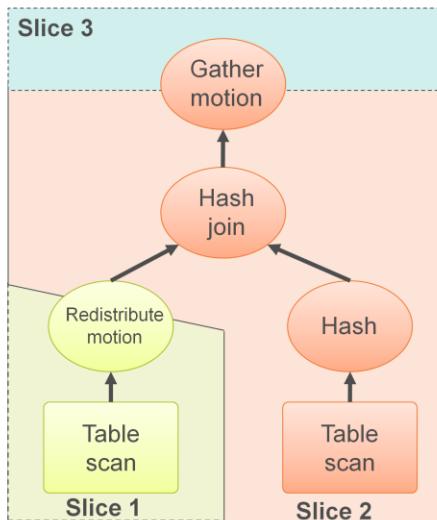
EXPLAIN ANALYZE is also an engineering tool used internally by Greenplum developers in the product development lifecycle. Query plans have information that assists the developers but also provides information from an end user perspective to analyze query execution and improve query performance.

Therefore, as an end user it is very important to focus on the query plan elements that are relevant to improving query performance.

## EXPLAIN Output

In query plans:

- Each node feeds its results to the node directly above
- There is one line for each node in the plan tree
- Each node represents a single operation
- Motion nodes are responsible for moving rows between segment instances



Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

81

Query plans are a right to left tree plan of nodes that are read from bottom to top with each node passing its result to the node directly above. There is one line for each node in the plan tree that represents a single operation, for example scan, join, aggregation or sort operation.

The node will identify the method used to perform the operation. For example a scan operation may perform a sequential scan or index scan. A join operation may perform a hash join or nested loop join.

The query plan will also include motion nodes. The motion operations are responsible for moving rows between segment instances required to process the query. The node will identify the method used to perform the motion operation for example, redistribute or broadcast motion.

A gather motion is when segments send the resulting rows to the master host. The last operation for most query plans will be a gather motion.

**Explain Output Tools**

**GREENPLUM COMMAND CENTER**

Welcome gadmin | About

Dashboard System Metrics Query Monitor Administration

« Back

Query ID: 1427832390-119533-4

Query Details Query Text Query Plan

Gather Motion 2:1 (slice3; segments: 2) (cost=145.34..437741.49 rows=13 width=6)
 > Hash Join (cost=145.34..437741.49 rows=7 width=6)
 Hash Cond: factontimeperformance.airlined = dimairline.airlineid
 > Hash Join (cost=122.03..437705.50 rows=2509 width=8)
 Hash Cond: factontimeperformance.originairportid = dimairport.airportid
 > Seq Scan on factontimeperformance (cost=0.00..385360.76 rows=10429488 width=12)
 > Hash (cost=121.98..121.98 rows=2 width=4)
 > Broadcast Motion 2:2 (slice1; segments: 2) (cost=0.00..121.98 rows=2 width=4)
 > Seq Scan on dimairline
 Filter: airlineid::text
 > Hash (cost=23.28..23.28 rows=1 width=2)
 > Broadcast Motion 2:2 (slice2; segments: 2) (cost=0.00..23.28 rows=1 width=2)
 > Seq Scan on dimairline (cost=0.00..23.28 rows=1 width=2)
 Filter: airlineid::text

Output pane

DATA OUTPUT Explain Message History

QUERY PLAN text

1 Gather Motion 2:1 (slice3; segments: 2) (cost=145.34..437741.49 rows=13 width=6)
 2 > Hash Join (cost=145.34..437741.49 rows=7 width=6)
 3 Hash Cond: factontimeperformance.airlined = dimairline.airlineid
 4 > Hash Join (cost=122.03..437705.50 rows=2509 width=8)
 5 Hash Cond: factontimeperformance.originairportid = dimairport.airportid
 6 > Seq Scan on factontimeperformance (cost=0.00..385360.76 rows=10429488 width=12)
 7 > Hash (cost=121.98..121.98 rows=2 width=4)
 8 > Broadcast Motion 2:2 (slice1; segments: 2) (cost=0.00..121.98 rows=2 width=4)
 9 > Seq Scan on dimairline (cost=0.00..121.98 rows=2 width=4)
 10 Filter: airlinedescription::text = 'Denver, CO: Denver International'::text
 11 > Hash (cost=23.28..23.28 rows=1 width=2)
 12 > Broadcast Motion 2:2 (slice2; segments: 2) (cost=0.00..23.28 rows=1 width=2)
 13 > Seq Scan on dimairline (cost=0.00..23.28 rows=1 width=2)
 14 Filter: airlineid::text = 'United Air Lines Inc.: UA'::text

Pivotal.

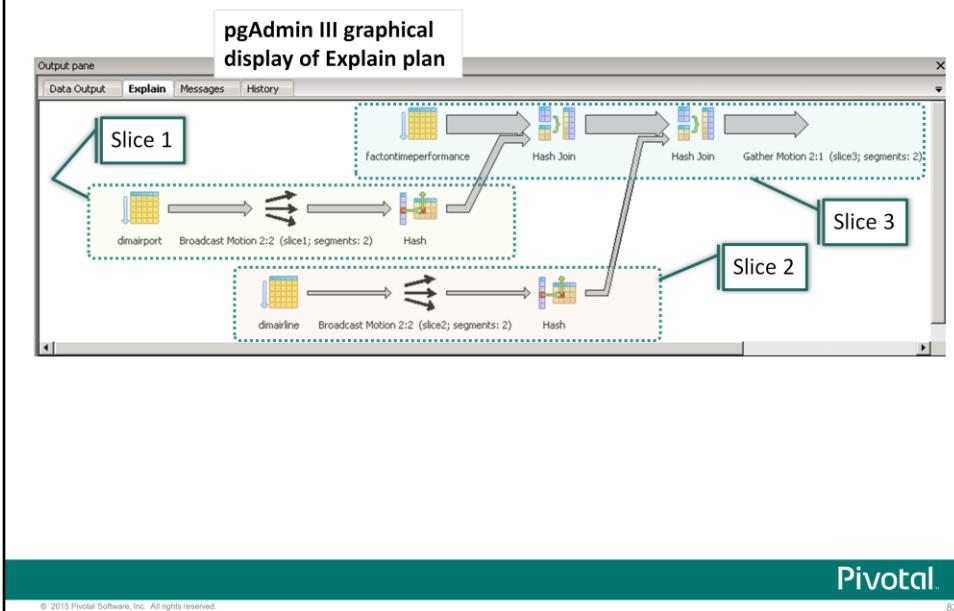
© 2015 Pivotal Software, Inc. All rights reserved.

Explain output can be rather lengthy and more difficult to read directly in PSQL. You can save the output to text files from PSQL by using the `\o <filename>` meta-command in PSQL.

You can view the explain plan in Greenplum Command Center by accessing the Query Monitor tab and clicking on the **Show Query Plan** button for an in-flight or completed query. The Query Plan button lets you view the full query details, the query text, and the query plan for the executing, completed, or aborted query.

pgAdmin III provides both a textual display and a graphical display. You click the **Explain query** button in pgAdmin III to view the plan analysis. While Command Center provides the explain plan in text for an in-flight query, pgAdmin III does not actually execute the query, but instead performs the same behavior as using the EXPLAIN syntax as part of the query. You do not need to prepend the EXPLAIN clause to the query to obtain this output.

## Graphical Display of Explain Plans



The graphical display shown here is similar to the graphical description provided earlier. Here, the plan is read from left to right and bottom to top. Slice 1 starts the operation with a scan on a table and performs a broadcast motion. In parallel, slice 2 starts a scan of another table and performs a broadcast motion of that table to the other segments to complete the operation. Once complete, the results are fed to another slice where another table is scanned and an hash join is performed between the results of slice 2 and slice 3. The results of slice 1 are hash joined with the results of slice 3 and a gather motion is then performed.

## EXPLAIN Example – Partition Elimination, Sorts, and Filters

Unique values are selected  
with the result grouped  
together



### Example: Query executed on a partitioned table

```
EXPLAIN SELECT DISTINCT (b.run_id), b.pack_id,
    b.local_time, b.session_id, b.domain
FROM display_run b
WHERE local_time >= '2007-03-01 00:00:00' AND
    local_time < '2007-04-01 00:00:00' AND (
        url='delete' OR url='estimate time' OR
        url='user time')
```

A sort is applied for the DISTINCT clause  
to guarantee a unique ordering of the rows

Filters are applied on columns on a  
partitioned table. Each partition will be scanned.

Pivotal

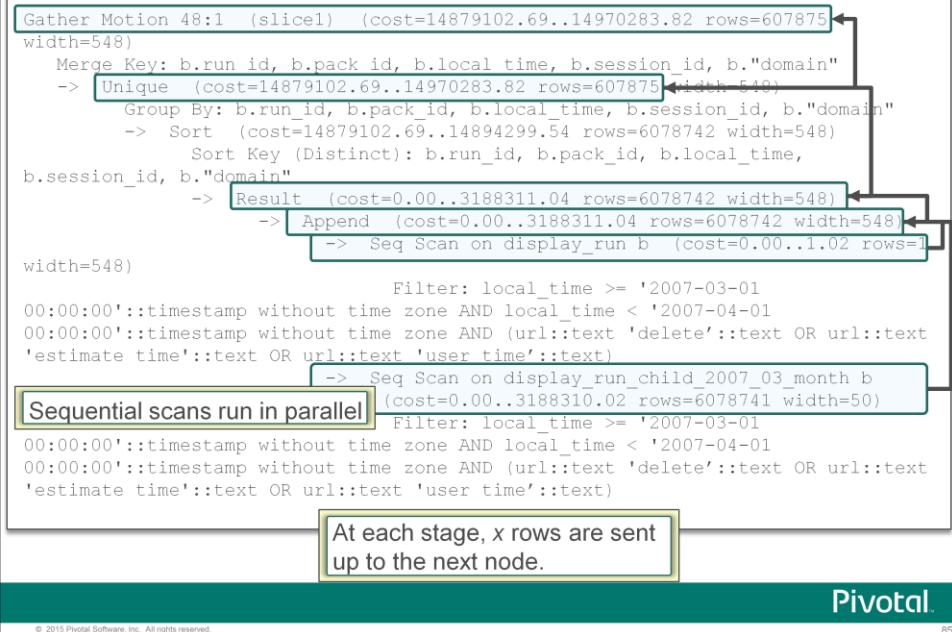
© 2015 Pivotal Software, Inc. All rights reserved.

84

Let us examine the query shown on the slide. The query shown on the slide was executed against a partitioned table, aliased b. There are several restrictions applied to the query, which includes a restriction on the `local_time` column and the `url` column.

The corresponding query plan is displayed on the next slide.

## EXPLAIN Example – Query Plan



The example query plan displayed consists of a tree plan with multiple nodes as indicated by the arrows symbol (->). Each node is executed in parallel across all segment instances. In the examples used, some elements of the plans may be omitted for readability.

As query plans are analyzed it is important to focus on the query plan elements that are relevant to improving query performance. Only those elements relevant for the end user will be discussed and analyzed.

Read the query plan from the bottom up:

- The query displayed begins with a sequential scan of the `display_run_child_2007_03_month` table.
- The WHERE clause is applied as a filter condition.
- The scan operation checks the condition for each row it scans, and outputs 6078741 rows that pass the condition.
- The sequential scan operation cost began at 0.00 to find the first row and 3188310.02 0.255 to scan all rows.
- A second sequential scan of the `display_run_b` table results in one row.
- The results of the scan operations are passed up to a sort operation which is required in order to perform the group operation.
- The output of the sort operation will pass 6078742 rows to the next plan node to perform a GROUP BY resulting in 607875 rows.
- Lastly a gather motion operation is performed where the segment instances send their rows to one of the segment instances.

## JOIN Order and Aggregation – Query



### Example: JOIN Operation on multiple tables

```
SELECT COUNT(*) FROM partsupp ps, supplier, part  
WHERE ps.ps_suppkey = supplier.s_suppkey  
AND part.p_partkey = ps.ps_partkey;
```

COUNT applies an aggregate over all of the columns.

JOIN operation is applied over two tables

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

86

In this example, multiple JOIN operations are performed on several tables. The result is an aggregation, in the form of a count, on the resulting rows.

Let us examine the query plan for this query.

## Begin by Examining Rows and Cost

```
Aggregate (cost=16141.02..16141.03 rows=1 width=0)
-> Gather Motion 2:1 (slice2) (cost=16140.97..16141.00 rows=1 width=0)
    -> Aggregate (cost=16140.97..16140.98 rows=1 width=0)
        -> Hash Join (cost=2999.46..15647.43 rows=197414 width=0)
            Hash Cond: ps.ps_partkey = part.p_partkey
            -> Hash Join (cost=240.46..9920.71 rows=200020 width=4)
                Hash Cond: ps.ps_suppkey = supplier.s_suppkey
                -> Seq Scan on partsupp ps (cost=0.00..6180.00)
                    rows=400000 width=8
                -> Hash (cost=177.97..177.97 rows=4999 width=4)
                    -> Broadcast Motion 2:2 (slice1)
                        (cost=0.00..177.97 rows=4999 width=4)
                        -> Seq Scan on supplier
                            (cost=0.00..77.99 rows=4999 width=4)
                            -> Hash (cost=1509.00..1509.00 rows=100000 width=4)
                                -> Seq Scan on part (cost=0.00..1509.00 rows=100000
width=4)
```



**Note:** Identify plan nodes where the estimated cost is very high and the number of rows are very large. This is where the majority of time is spent.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

87

To analyze query plans and identify opportunities for query improvement:

- Begin by examining the rows and cost for the plan nodes.
- Identify plan nodes where the estimated number of rows are very large.
- Identify plan nodes where the estimated cost to perform the operation is very high.
- Determine if the estimated number of rows seem reasonable and the cost relative to the number of rows for the operation performed.
- Ensure that database statistics are up to date. High quality, up to date database statistics is required for the query planner's estimates.

In the example displayed:

- A sequential scan of the `supplier` table is performed.
- This is followed by a sequential scan of the `partsupp` table.
- To perform the hash join of the `supplier` and `partsupp` tables, a broadcast of the `supplier` table is performed. The broadcast motion notion of 2:2 indicates that there are two segment instances that will broadcast its rows in the `supplier` table to the other segment instances.
- The final gather motion is when segment instances send its rows to the master host. In this example there are two segment instances sending to one master instance (2 : 1).

## Validate Partition Elimination

```
Gather Motion 48:1  (slice1)  (cost=174933650.92..176041040.58 rows=7382598 width=548)
  Merge Key: b.run_id, b.pack_id, b.local_time, b.session_id, b."domain"
    -> Unique (cost=174933650.92..176041040.58 rows=7382598 width=548)
      Group By: b.run_id, b.pack_id, b.local_time, b.session_id, b."domain"
        -> Sort (cost=174933650.92..175118215.86 rows=73825977 width=548)
          Sort Key (Distinct): b.run_id, b.pack_id, b.local_time,
          b.session_id, b."domain"
            -> Result (cost=0.00..31620003.26 rows=73825977 width=548)
              -> Append (cost=0.00..31620003.26 rows=73825977 width=548)
                -> Seq Scan on display_run b (cost=0.00..1.02 rows=1
width=548)
                  Filter: url::text 'delete'::text OR url::text
'estimate time'::text OR url::text 'user time'::text
                  -> Seq Scan on display_run_child_2007_03_month
b (cost=0.00..2635000.02 rows=6079950 width=50)
                  Filter: url::text 'delete'::text OR url::text
'estimate time'::text OR url::text 'user time'::text
                  -> Seq Scan on display_run_child_2007_04_month
b (cost=0.00..2635000.02 rows=6182099 width=50)
                  Filter: url::text 'delete'::text OR url::text
'estimate time'::text OR url::text 'user time'::text
...

```

All child partitions are scanned

Pivotal.

88

If using table partitioning, it is important to validate that partitions are being eliminated or pruned. Only the partitions that contain data to satisfy the query should be scanned. The EXPLAIN ANALYZE output will display the child partitions that will be scanned.

In the example displayed, all ten child partitions will be scanned to satisfy the query. Note that only two partitions are displayed for brevity. A sequential scan is performed for the display\_run parent table, but keep in mind the parent table contains no data.

It is important to remember that to eliminate partitions, the partitioning criteria, CHECK clause, for the child tables is the same criteria used in the query predicate, WHERE clause. If the query access (WHERE clause) does not match the partitioning definition, the benefit of partition elimination is not achieved.

In the example displayed, the display\_run table was partitioned on local\_time and was not used in the query predicate.

The SQL for the query displayed is as follows:

```
SELECT DISTINCT run_id, pack_id, local_time, session_id,
domain
FROM display_run b
WHERE (url LIKE '%.delete%' OR
      url LIKE '%.estimated time%' OR
      url LIKE '%.user time%' );
```

## Partition Elimination

```
Gather Motion 48:1 (slice1) (cost=14879102.69..14970283.82 rows=607875 width=548)
  Merge Key: b.run_id, b.pack_id, b.local_time, b.session_id, b."domain"
    -> Unique (cost=14879102.69..14970283.82 rows=607875 width=548)
      Group By: b.run_id, b.pack_id, b.local_time, b.session_id, b."domain"
        -> Sort (cost=14879102.69..14894299.54 rows=6078742 width=548)
          Sort Key (Distinct): b.run_id, b.pack_id, b.local_time,
          b.session_id, b."domain"
            -> Result (cost=0.00..3188311.04 rows=6078742 width=548)
              -> Append (cost=0.00..3188311.04 rows=6078742 width=548)
                -> Seq Scan on display_run b (cost=0.00..1.02 rows=1
width=548)
                  Filter: local_time >= '2007-03-01
00:00:00'::timestamp without time zone AND local_time < '2007-04-01
00:00:00'::timestamp without time zone AND (url::text 'delete'::text OR url::text
'estimate time'::text OR url::text 'user time'::text)
                    -> Seq Scan on display_run_child_2007_03_month b
                      (cost=0.00..3188310.02 rows=6078741 width=50)
                        Filter: local_time >= '2007-03-01
00:00:00'::timestamp without time zone AND local_time < '2007-04-01
00:00:00'::timestamp without time zone AND (url::text 'delete'::text OR url::text
'estimate time'::text OR url::text 'user time'::text)
```

Partition is on local\_time. WHERE clause is on local\_time. Partition elimination is achieved.

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

89

To achieve partition elimination, the query predicate must be the same as the partitioning criteria.

It is also important to note that to eliminate partitions, the WHERE clause must contain an explicit value. The WHERE clause can not contain a subquery if partitions are to be eliminated.

For example, WHERE date = '1/30/2008' is an explicit value. However, WHERE date IN (SELECT.... WHERE) is an example of a subquery.

Building upon the same example, the display\_run table was partitioned on local\_time and the WHERE clause contained an explicit value that eliminated all but one partition. This partition contained the data required to satisfy the query.

The SQL for the query displayed is as follows:

```
SELECT DISTINCT run_id, pack_id,
local_time, session_id, domain
FROM display_run b
WHERE (b.local_time >= '2007-03-01 00:00:00'::timestamp
without time zone AND b.local_time < '2007-04-01
00:00:00'::timestamp without time zone)
AND (url LIKE '%.delete%' OR url LIKE '%.estimated time%'
OR url LIKE '%.user time%' );
```

## Partition Elimination (Cont)



**Note:** To eliminate partitions, the WHERE clause must be the same as the partition criteria AND must contain an explicit value (no subquery).

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

90

## Partition Elimination (Continued)

## Optimal Plan Heuristics

When analyzing query plans, design queries that select from the top set of bullets over the bottom set of bullets:

Faster Operators	Slower Operators
Sequential Scan	
Hash JOIN	Nested Loop JOIN Merge JOIN
Hash Aggregate	Sort
Redistribute Motion	Broadcast Motion

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

91

The second set of bullets shown on the slide represents actions that are considered slower than the top set of bullets.

The EXPLAIN ANALYZE contains one line for each node in the plan tree. That line represents a single operation, such as scan, join, aggregation, or sort operation.

The node will identify the method used to perform the operation. For example:

- A scan operation may perform a sequential scan or index scan.
- A join operation may perform a hash join or nested loop join. A sort operation may perform a hash aggregation or sort.

When analyzing query plans, identify the nodes with slow operators including nested loop join, merge join, sort and broadcast motion operators. Begin by examining the rows and cost for the plan nodes.

For these plan nodes, review the estimated number of rows and cost. If the estimated number of rows are large and the cost to perform the operation is high, then there are opportunities to improve query performance.

A sequential scan in an MPP environment is an efficient method of reading data from disk as each segment instance works independently in the shared nothing environment. This is unlike a traditional SMP environment where queries compete for resources. Data warehouse environments typically access large volumes of data for analysis. However, an index scan may be beneficial when the selectivity of queries are high accessing a single or few rows.

## Nested Loops and Broadcasts



### Example: Nested loop query

```
SELECT * FROM factontimeperformance f1,  
factontimeperformance2 f2  
WHERE f1.originairportid~*'JAX' AND  
f2.quarterid=2;
```

Outer table is scanned and  
the filter is applied over each row

Broadcast is performed on  
the table determined by  
Greenplum to be smaller

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

92

In this example, a nested loop performs a scan on the outer table and iterates over each row of the outer table, applying a filter, if necessary, to those rows.

The smaller table is broadcast across segments, if the JOIN is not on the distribution key of both tables.

## Eliminate Nested Loop Joins

```
Gather Motion 2:1 (slice2; segments: 2) (cost=523433.54..585132830423.58
rows=4078219329455 width=1015)
  -> [Nested Loop (cost=523433.54..585132830423.58 rows=4078219329455
width=1015)
    -> Broadcast Motion 2:2 (slice1; segments: 2) (cost=0.00..486388.79
rows=1617783 width=524)
      -> Seq Scan on factontimeperformance f (cost=0.00..437855.33
rows=808892 width=524)
        Filter: originairportid ~* 'JAX'::text
      -> Materialize (cost=523433.54..653859.95 rows=2520871 width=491)
        -> Seq Scan on factontimeperformance2 d (cost=0.00..438382.80
rows=2520871 width=491)
        Filter: quarterid = 2
```



**Note:** Eliminate nested loop joins for hash joins.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

93

A nested loop join requires:

- Each outer tuple is compared with each inner tuple that might join to it.
- The broadcast of one of the tables so that all rows in one table can be compared to all rows in the other table. The performance is therefore affected by the size of the tables.

Nested loop joins may perform well for small tables. There are performance implications when joining two large tables.

In the example displayed, the estimated cost to perform the join is very high relative to the estimated number of rows.

For plan nodes that contain a nested loop join operator, validate the SQL and ensure that the results are what is intended. Poorly written or incorrect SQL is often times the primary offender affecting query performance.

In addition, the `enable_nestloop` system configuration parameter should be set to `off` to favor a hash join over a nested loop join when possible. By default, the parameter is set to `on`.

## Replace Large Sorts

```
Redistribute Motion 48:48  (slice2)  (cost=8568413851.73..8683017216.78
rows=83317 width=154)
-> GroupAggregate  (cost=8568413851.73..8683017216.78 rows=83317 width=154)
  Group By: b."host", "customer_id"
  -> Sort  (cost=8568413851.73..8591334233.13 rows=9168152560 width=154)
    Sort Key: b."host", "customer_id"
    -> Redistribute Motion
48:48  (slice1)  (cost=2754445.61..1748783807.47 rows=9168152560 width=154)
  Hash Key: b."host", "customer_id"
  -> Hash Join  (cost=2754445.61..1565420756.27
rows=9168152560 width=154)
    Hash Cond: a.pack_id = b.pack_id
    Join Filter: (a.local_time - b.local_time) >= '-00:10:00'::
```



**Note:** Replace large sorts with HashAgg.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

94

Identify plan nodes where a sort or aggregate operation is performed. Hidden inside an aggregate operation is sort. If the sort or aggregate operation involves a large number of rows, there is opportunity to improve query performance. A HashAggregate operation is preferred over sort and aggregate operations when a large number of rows are required to be sorted.

Usually, a sort operation is chosen by the optimizer due to the SQL construct, that is, due to the way the SQL is written. Most sort operations can be replaced with a HashAgg if the query is rewritten.

## Re-write Sort Example

The optimizer:

- Dynamically re-writes a single count distinct to replace a sort for a HashAgg by pushing the distinct into a subquery. For example:

```
SELECT count( distinct l.quantity ) FROM  
lineitem;  
  
SELECT count(*) FROM (SELECT l.quantity  
FROM lineitem  
GROUP BY l.quantity) as foo;
```

- Can not rewrite multiple count distincts

Correct the query with the following:

- Push the distincts into a subquery that inserts the rows into a temp table
- Query the table with the distinct values to obtain the count

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

95

A sort is performed when the SQL involves a count distinct SQL construct.

The optimizer will dynamically re-write a query that contains a count distinct to obtain a HashAgg operation instead of a sort operation. The optimizer re-write basically pushes the distinct into a subquery.

However, the optimizer can not re-write a query that contains multiple count distincts so a sort operation will be performed.

To replace the sort operation for a HashAgg operation the query will need to be rewritten. The re-write is similar to the example displayed, except a temp table will be used to store the distinct values.

In this scenario, re-write the query to push the distinct into a subquery that inserts the resulting rows into a temp table. Then select from the temp table that contains the distinct values to obtain the count.

## Eliminate Large Table Broadcast Motion

```
-> Hash (cost=18039.92..18039.92 rows=20 width=66)
   -> Redistribute Motion
24:24 (slice3) (cost=0.55..18039.92 rows=20 width=66)
   Hash Key:
   cust_contact_activity.src_system_id::text
      -> Hash Join (cost=0.55..18039.52 rows=20
width=66)
         Hash Cond:
         cust_contact_activity.contact_id::text = cust_contact.contact_id::text
            -> Seq Scan on
         cust_contact_activity (cost=0.00..15953.63 rows=833663
width=39)
      A small table broadcast is acceptable.
      Hash (cost=0.25..0.25 rows=24
width=84)
      -> Broadcast Motion 24:24 (slice2)
         (cost=0.00..0.25 rows=24 width=84)
         -> Seq Scan on
         cust_contact (cost=0.00..0.00 rows=1 width=84)
```



**Note:** Use `gp_segments_for_planner` to increase the cost of the motion to favor a redistribute motion.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

96

In some cases, a broadcast motion will be performed rather than a redistribute motion.

In a broadcast motion, every segment instance performs a broadcast of its own individual rows to all other segment instances. This results in every segment instance having its own complete and local copy of the entire table.

A broadcast motion may not be as optimal as a redistribute motion. A broadcast motion is not acceptable for large tables. When analyzing query plans, identify broadcast motion nodes. If the number of rows are large, then consider using the `gp_segments_for_planner` parameter to increase the cost of the motion.

The `gp_segments_for_planner` sets the number of primary segment instances for the planner to assume in its cost and size estimates. If `gp_segments_for_planner` is set to zero, then the value used is the actual number of primary segments.

This variable affects the planner's estimates of the number of rows handled by each sending and receiving process in motion operators. Increasing the number of primary segments increases the cost of the motion, thereby favoring a redistribute motion over a broadcast motion. For example setting `gp_segments_for_planner = 100000` tells the planner that there are 100000 segments.

## Increase work\_mem Parameter

```
-> HashAggregate (cost=74852.40..84739.94 rows=791003 width=45)
   Group By: l_orderkey, l_partkey, l_comment
   Rows out: 2999671 rows (seg1) with 13345 ms to first row, 71558 ms to
   end, start offset by 3.533 ms.
   Executor memory: 2645K bytes avg, 5019K bytes max (seg1).
   Work_mem used: 2321K bytes avg, 4062K bytes max (seg1).
   Work_mem wanted: 237859K bytes avg, 237859K bytes max (seg1) to lessen
workfile I/O
   affecting 1 workers.

   .
   .
-> Seq Scan on lineitem (cost=0.00..44855.70 rows=2999670 width=45)
   Rows out: 2999671 rows (seg1) with 0.571 ms to first row, 4167 ms to
   end, start offset by 4.105
Slice statistics:
(slice0) Executor memory: 211K bytes.
(slice1) * Executor memory: 2840K bytes avg x 2 workers, 5209K bytes max (seg1).
Work_mem: 4062K bytes max, 237859K bytes wanted.
Settings: work_mem=4MB
Total runtime: 73326.082 ms
(24 rows)
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

97

Identify plan nodes where the memory used versus the memory wanted for the specified operation is significant. At the bottom of the plan file, the `work_mem` parameter setting is also displayed.

The `work_mem` variable specifies the amount of memory to be used by internal sort operations and hash tables before spilling to file system. If there are several sort or hash operations running in parallel for a given query, each operation will be permitted to use the memory specified before spilling to file system.

If the memory used versus the memory wanted for a plan node operation is significant, then increase `work_mem` for the query using the SQL SET statement to give the query additional memory.

Arbitrarily setting `work_mem` to a very large number for a given query may:

- Decrease query performance due to the overhead of allocating and managing memory.
- Impact on other concurrent queries with sort and hash operations.

## Monitoring and Terminating Runaway Queries

The screenshot shows a PostgreSQL terminal window with two queries. The first query selects from the session\_state.session\_level\_memory\_consumption view where current\_query is not 'IDLE'. The second query counts rows in faadata factontimeperformance grouped by gp\_segment\_id.

Segment ID	Current vmem (MB) for the session on the segment	Is the session a runaway on the segment	Number of query processes for the session	Active query processes for the session	Incomplete query processes	Vmem memory when the session was marked as a runaway	Number of commands when the session was marked as a runaway
faal	69409	gpadmin	10	10	0	0	0
faal	69409	gpadmin	2	2	-1	0	0

**Output of the session\_state.session\_level\_memory\_consumption view**

runaway\_detector\_activation\_percent:  
• Determines when queries are terminated  
• Can be disabled by setting it 0.

Pivotal.  
© 2015 Pivotal Software, Inc. All rights reserved. 98

While manipulating the `work_mem` parameter can work for individual queries, the workload resource management tool lets you manage queries for other users. If not defined appropriately, a query can potentially consume large amounts of memory and block other sessions from executing their queries.

The `session_level_memory_consumption` view, defined in the `session_state` schema, provides information on the amount of memory that a session is using to execute its queries. The view provides a view of the query consumption at the segment level for the associated query. The segment listing includes the master.

The view is installed per database using the script,  
`$GPHOME/share/postgresql/contrib/gp_session_state.sql`.

In addition to providing information on the amount of memory the query is consuming with the `vmem_mb` field, the `is_runaway` field provides an indication of whether or not Greenplum considers the session a runaway. A session is considered a runaway when the amount of vmem memory it is consuming exceeds the value defined in the `runaway_detector_activation_percent` parameter. Greenplum will terminate the query, starting with the largest consumer. By default, this parameter is set to 90%. It can be disabled by changing the value to 0.

## Workfiles (Spill Files)

Consider the following:

- Operations performed in memory are optimal
- Insufficient memory means rows will be written out to disk as spill files
- There is a certain amount of overhead with any disk I/O operation

Spill files:

- Are located within the `pgsql_tmp` directory for the database
- Indicate the node operation

```
[jesh:~/gpdb-data/seg1/base/16571/pgsql_tmp] ls -ltrh
total 334464
-rw----- 1 jeshle jeshle 163M Jan 9 23:41
pgsql_tmp_SortTape_Slice1_14022.205
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

99

Plan node operations performed in memory is optimal for query performance. Join, aggregate, and sort operations that can not be performed in memory will impact query performance.

For example, a hash join performed in memory is the optimal method for joining tables. However, if the build step of a hash join involves a very large number of rows, there may not be sufficient memory. In this scenario the build rows will be written out to disk as spill files. With any disk I/O operation there is a certain amount of overhead.

Inside the directory, `/data_directory/base`, you will find several directories whose names represent the OIDs of the databases within a segment instance. Within any of these directories, there is a `pgsql_temp` directory where the spill files are located. The filenames will indicate the node operation.

## Workfile Improvements – Management

- Improvements have been made in the 4.2.5 and above release to workfiles (also known as spill files).
- There has historically been no supported way to manage the size of these workfiles or to understand what workfiles exist in previous versions.
- There are new parameter options to limit the size of workfiles created:

gp_workfile_limit_per_query	Limits the amount of workfile space that any particular query can use; protects against excessive workfile sizes
gp_workfile_limit_per_segment	Limits the amount of workfile space that can be used on any particular segment server; Protects against “out of disk space” errors
gp_workfile_compress_algorithm	Compression algorithm to use on spill files generated by hash aggregation or hash join operations

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

100

These following parameters help to control how large workfiles can get for queries that will generate them and how to handle the size of these spill files. This effectively helps you to control queries that process very large data sets that cannot be performed totally in memory. This can be used to help protect against bad user queries that use excessive workfile sizes and can also be used to help protect against “out of disk space” conditions.

The workfile parameters introduced are:

- `gp_workfile_limit_per_query` – This sets the maximum disk size (in kilobytes) an individual query is allowed to use for creating temporary spill files at each segment. The default value is 0, which means a limit is not enforced. This parameter can be used to protect the environment against queries that can consume excessive workfile sizes for execution.
- `gp_workfile_limit_per_segment` – Sets the maximum total disk size (in kilobytes) that all running queries are allowed to use for creating temporary spill files at each segment. The default value is 0, which means a limit is not enforced.
- `gp_workfile_compress_algorithm` – This parameter is used to define the algorithm that can be used to compress spill files generated by either a hash aggregation or hash join. It can be set to `zlib` or disabled with `none`, the default setting.

## Workfile Improvements – Management Views

View	Description
gp_workfile_entries	Lists individual workfiles. The view contains one row for each operator using disk space for workfiles on a segment at the current time
gp_workfile_usage_per_query	Rollup of workfiles per query. The view contains one row for each query using disk space for workfiles on a segment at the current time.
gp_workfile_usage_per_segment	Rollup of workfiles per segment. The view contains one row for each segment. Each row displays the total amount of disk space used for workfiles on the segment at the current time.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

101

The gp\_workfile\* views are available in the gp\_toolkit schema. These views are:

- gp\_workfile\_entries – This view contains one row for each operator that uses disk space for workfiles at the current time. If an operator uses disk space for workfiles on multiple segments, the view contains one row for each segment where that occurs. If multiple operators in a query use disk space for workfiles, the view contains one row for each operator using disk space for workfiles for each segment where that occurs.
- gp\_workfile\_usage\_per\_query – This view contains one row for each query that uses disk space for workfiles at the current time. If a query uses disk space for workfiles on multiple segments, the view contains one row for each segment where that occurs.
- gp\_workfile\_usage\_per\_segment – This view contains one row for each segment. Each row displays the total amount of disk space used for workfiles on the segment at the current time.

This information can be used for troubleshooting and tuning queries. The information in the views can also be used to specify the values for the Greenplum Database configuration parameters `gp_workfile_limit_per_query` and `gp_workfile_limit_per_segment`.

## Identify Respill in Hash Agg Operations

```
...  
    -> HashAggregate (cost=74852.40..84739.94 rows=791003 width=45)  
        Group By: l_orderkey, l_partkey, l_comment  
        Rows out: 2999671 rows (seg1) with 13345 ms to first row, 71558 ms to end . .  
.  
    Executor memory: 2645K bytes avg, 5019K bytes max (seg1).  
    Work_mem used: 2321K bytes avg, 4062K bytes max (seg1).  
    Work_mem wanted: 237859K bytes avg, 237859K bytes max (seg1) to lessen  
    workfile I/O  
    affecting 1 workers.  
    (seg1) 2999671 groups total in 5 batches; 64 respill passes; 23343536 respill  
    rows.  
    (seg1) Initial pass: 44020 groups made from 44020 rows; 2955651 rows spilled  
    to workfile.  
    (seg1) Hash chain length 5.0 avg, 18 max, using 602986 of 607476 buckets.  
-> Seq Scan on lineitem (cost=0.00..44855.70 rows=2999670 width=45)  
    Rows out: 2999671 rows (seg1) with 0.571 ms to first row, 4167 ms to end,  
    start offset by 4.105  
Slice statistics:  
(slice0) Executor memory: 211K bytes.  
(slice1) * Executor memory: 2840K bytes avg x 2 workers, 5209K bytes max (seg1).  
Work_mem: 4062K bytes max, 237859K bytes wanted.  
Settings: work_mem=4MB
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

102

A HashAgg attempts to build the hash table in memory. If the hash table size exceeds the available memory, it will start to spill, that is, write rows out to temporary files, called batches.

Later, the HashAgg will process batch by batch. If the batch still cannot fit in memory, each batch will spill to more batches. This is respill condition.

Identify HashAggregate node plans with respill conditions. If the number of respill passes and the number of respill rows are large, than there is opportunity to increase query performance.

In order to minimize respilling, the HashAggregate node uses the planners estimates to subdivide the data into batches and will spill a separate data file per batch.

If the estimated number of batches is smaller than the real optimal value, then respilling of data will be required when processing batches. This respilling behavior can be quite expensive if the difference between the estimated and optimal batch value is great.

## Review JOIN Order – Query Plan

```
Aggregate (cost=16141.02..16141.03 rows=1 width=0)
-> Gather Motion 2:1 (slice2) (cost=16140.97..16141.00 rows=1 width=0)
    -> Aggregate (cost=16140.97..16140.98 rows=1 width=0)
        -> Hash Join (cost=2999.46..15647.43 rows=197414 width=0)
            Hash Cond: ps.ps_partkey = part.p_partkey
            -> Hash Join (cost=240.46..9920.71 rows=200020 width=4)
                Hash Cond: ps.ps_suppkey = supplier.s_suppkey
                -> Seq Scan on partsupp ps (cost=0.00..6180.00
rows=400000 width=8)
                -> Hash (cost=177.97..177.97 rows=4999 width=4)
                    -> Broadcast Motion 2:2 (slice1)
(cost=0.00..177.97 rows=4999
width=4)
                -> Seq Scan on supplier
(cost=0.00..77.99 rows=4999 width=4)
                    -> Hash (cost=1509.00..1509.00 rows=100000 width=4)
                        -> Seq Scan on part (cost=0.00..1509.00
rows=100000 width=4)
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

103

For a join, the execution order to build on the smaller tables or hash join result and probe with larger tables.

Review:

- The execution order of the query plan tree.
- The estimated number of rows.

Optimally the largest table is used for the final join or probe to reduce the number of rows being passed up the tree to the topmost plan nodes.

If the analysis reveals that the order of execution builds and/or probes on the largest table first ensure that database statistics are up to date.

## Use `join_collapse_limit` to Specify Join Order

To specify the join order:

- Set the parameter, `join_collapse_limit=1`
- Use ANSI style join as in the following example

```
SELECT COUNT(*) FROM partsupp ps
    JOIN supplier ON ps_suppkey = s_suppkey
    JOIN part ON p_partkey = ps_partkey;
```

The following is an example of a non-ANSI style join:

```
SELECT COUNT(*) FROM partsupp, supplier, part
    WHERE ps_suppkey = s_suppkey
    AND p_partkey = ps_partkey;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

104

If the analysis reveals that the order of execution builds and/or probes on the largest table first and database statistics are up to date, consider using the Global User Configuration (GUC) variable, also known as the configuration parameter, `join_collapse_limit`, to specify the join order for ANSI style joins. This parameter will not affect non-ANSI style joins.

Set `join_collapse_limit = 1` to disable the join ordering as determined by the optimizer. It is then the user's responsibility to write the query in a way, using ANSI style joins, that sets the preferred join ordering.

## Analyzing Query Plans Summary

The following summarizes how to analyze a query plan:

- Identify plan nodes with a large number of rows and high cost
- Validate partitions are being eliminated
- Eliminate large table broadcast motion
- Replace slow operators in favor of fast operators
- Identify spill files and increase memory
- Identify respill in HashAggregate
- Review join order

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

105

Analyzing EXPLAIN plan files is the best possible way to understand query execution and query performance:

- Begin by reviewing the plan nodes where there majority of time is spent in executing the query. For these plan nodes review the estimated number of rows and cost. Does the estimated number of rows and relative cost seem reasonable? If not, validate database statistics are up to date.
- Validate partitions are being eliminated. Only the partitions that contain data to satisfy the query should be scanned.
- Eliminate large table broadcast motion for redistribute motion.
- Replace slow operators for fast operators.
- Identify plan nodes where the memory used versus the memory wanted for the specified operation is significant. If the memory used versus the memory wanted for a plan node operation is significant then increase `work_mem` for the query.
- Review the execution order of the query plan tree. If the analysis reveals that the order of execution builds and/or probes on the largest table first ensure that database statistics are up to date.
- And most importantly review the SQL and ensure that the results are what is intended. Poorly written or incorrect SQL is often times the primary offender affecting query performance.

## Lab: Explain the Explain Plan – Analyzing Queries

In this lab, you will analyze a set of queries provided to you.

You will:

- Analyze queries

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

106

In this lab, you will analyze a set of queries provided to you.

## Module 8: Performance Analysis and Tuning

### Lesson 4: Summary

During this lesson the following topics were covered:

- The benefits of using the Greenplum EXPLAIN and EXPLAIN ANALYZE utilities
- Deciphering an explain plan based on output provided
- Analyzing the query plan

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

107

This lesson focused on the query plan and how to use the EXPLAIN and EXPLAIN ANALYZE syntax to not only view the query plan, but, as in the latter syntax, improve overall statistics and the performance of the query based on the optimizer's ability to more accurately predict what is needed. A description of the explain plan was provided along with examples of major nodes and operations to look for when analyzing the query plan.

# Module 8: Performance Analysis and Tuning

## Lesson 5: Improve Performance with Statistics

In this lesson, you examine how to use improve performance for the database by keeping statistical information up to date.

Upon completion of this lesson, you should be able to:

- Collect statistics about the database
- Improve performance by using ANALYZE
- Increase sampling and statistics
- Identify when it is best to use ANALYZE

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

108

Keeping statistical information up on tables to date directly impacts how the optimizer creates the query plan.

In this less, you will:

- Collect statistics about the database
- Improve performance by using ANALYZE
- Increase sampling and statistics
- Identify when it is best to use ANALYZE

## EXPLAIN ANALYZE Estimated Costs

EXPLAIN ANALYZE provides cost estimates for the execution of the plan node as follows:

- Cost – Measured in units of disk page fetches
- Rows – The number of rows output by the plan node
- Width – Total bytes of all the widest row of the rows output by the plan node

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

109

Query plans display the node type along with the planner estimates including cost, rows and width for the execution of that plan node.

Cost is a relative measured in units of disk page fetches. The value 1.0 equals one sequential disk page read. The first estimate is the cost of getting the first row and the second is the total cost of getting all rows in milliseconds. The total cost assumes that all rows will be retrieved, which may not always be the case if using a LIMIT clause for example.

Rows are the total number of rows output by the plan node. This is usually less than the actual number of rows processed or scanned by the plan node, reflecting the estimated selectivity of any WHERE clause conditions.

The width is the total bytes of the widest row of all the rows output by the plan node.

It is important to note that the cost of an upper-level node in the plan tree includes the cost of all its child nodes. The topmost node of the plan has the estimated total execution cost for the plan.

The cost does not consider the time spent transmitting results to the client.

## ANALYZE and Database Statistics

Database statistics:

- Are used by the optimizer and query planner
- Should be updated with ANALYZE:
  - After loading data
  - After large INSERT, UPDATE, and DELETE operations
  - After CREATE INDEX operations
  - After database restores from backups

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

110

The Greenplum database uses a cost-based query planner that relies on database statistics. The ANALYZE command collects statistics about the database needed by the query planner to determine the most efficient way to execute a query.

It is very important that the database has up-to-date statistics for optimal query performance. As a database or system administrator this is the most important ongoing administrative database maintenance task.

It is highly recommended to run ANALYZE:

- After data loads
- After large INSERT, UPDATE, and DELETE operations
- After CREATE INDEX operations
- After completing a database restore from backups

The `pg_statistic` system catalog table stores statistical data about the contents of the database. Entries are created by ANALYZE and subsequently used by the query planner. There is one entry for each table column that has been analyzed. The `pg_statistic` system catalog also stores statistical data about the values of index expressions.

## ANALYZE and VACUUM ANALYZE

### The ANALYZE command:

- Is used to generate database statistics
- Can be used on specific table and column names  
`ANALYZE [table [ (column [, ...] ) ]]`

### The VACUUM ANALYZE command:

- Is used to vacuum the database
- Generates database statistics
- Can be used on specific table and column names  
`VACUUM ANALYZE [table [(column [, ...] )]]`

Pivotal.

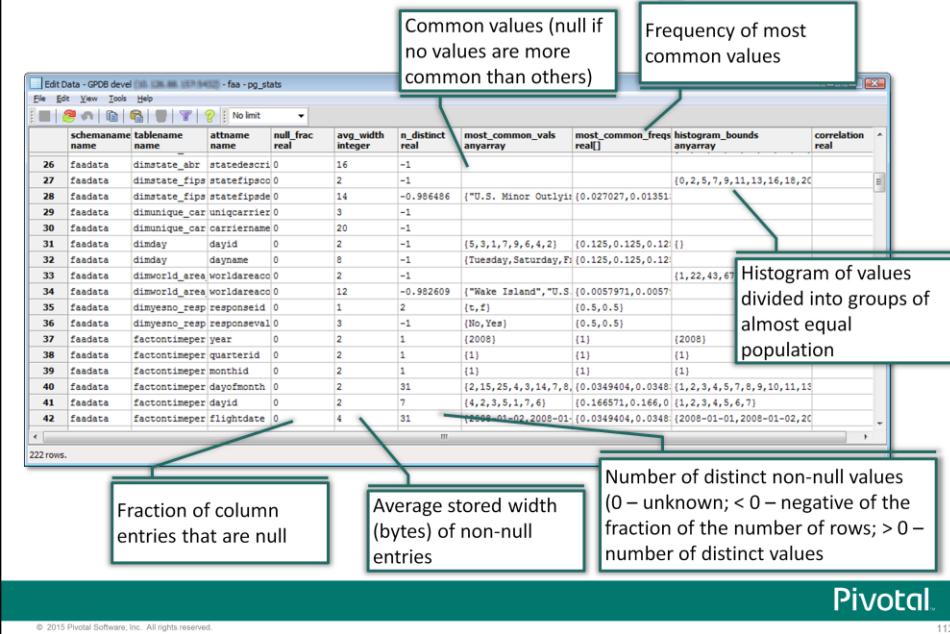
© 2015 Pivotal Software, Inc. All rights reserved.

111

Use the ANALYZE command to generate database statistics. An ANALYZE may be run on the entire table or on specified columns in a table. In large data warehouse environments it may not be feasible to run ANALYZE on an entire database or table due to time constraints. Specifying particular columns to analyze provides the flexibility to generate optimal database statistics for large databases.

Use VACUUM ANALYZE to vacuum the database and generate database statistics. VACUUM ANALYZE performs a VACUUM and then an ANALYZE for each selected table. This is a convenient combination for routine database maintenance scripts.

## Storage of Statistical Data in pg\_stats



Statistical information used by the optimizer is stored in the pg\_catalog.pg\_statistic table. This table stores statistical data for each entry of a table column that has been analyzed with the ANALYZE command. The pg\_statistic table is accessible to database superusers as it provides information on all tables within the database. The pg\_stats view is a publicly readable view which gives the current user information on tables they have access to. This protects sensitive information by only displaying information on tables the user has read access to.

The statistical columnar data is as follows:

- null\_frac\_real – This column displays the fraction of column entries that are null.
- avg\_width – The average width of the non-null values of the column is displayed.

## **Storage of Statistical Data in pg\_stats**

- `n_distinct` – The number of distinct non-null values is displayed. If the value displayed is 0, the number of distinct values is unknown. If it is less than 0, the value displayed is the negative of the number distinct values divided by the number of rows in the table. The negative value indicates that `ANALYZE` believes the number of distinct values will increase as the number of rows increases. A positive value indicates the number of possible values for the column.
- `most_common_vals` – The list of most common values is displayed in this column. This column is null if there are no values which are more common than other values.
- `most_common_freqs` – The frequency of the most common values is displayed. This represents the number of occurrences of each divided by the number of rows. The value is null if `most_common_vals` is null.
- `histogram_bounds` – The column displays the histogram of the list of values that divide the values into groups of approximately equal population. This does not include the values in the `most_common_vals` column, if it is populated. If the `histogram_bounds` field is null, this is likely a result of the `most_common_vals` field representing the entire population.

## Use SET STATISTICS to Increase Sampling

Sampling for statistics:

- Can be increased for a given column with `ALTER TABLE ... SET STATISTICS`
- Defaults to 25
- May improve query planner estimates for columns used in query predicates and joins (`WHERE` clause)

```
ALTER TABLE customer ALTER customer_id  
SET STATISTICS 35;
```

- Can impact the time it takes to `ANALYZE` if statistics has larger values

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

114

For large tables, `ANALYZE` takes a random sample of the table contents, rather than examining every row to allow large tables to be analyzed in a small amount of time. To increase sampling and statistics collected by `ANALYZE` use the `ALTER TABLE...SET STATISTICS` command.

`SET STATISTICS` can be used to alter the statistics value for a particular table column. This is recommended for columns frequently used in query predicates and joins. The default statistics value is 25. This default value is set with the `default_statistics_target` parameter.

Setting the statistics value to zero disables collection of statistics for that column. It may be useful to set the statistics value to zero for columns that are never used as part of the `WHERE`, `GROUP BY`, or `ORDER BY` clauses of queries, since the query planner has no use for statistics on such columns.

Larger values increase the time needed to do `ANALYZE`, but may improve the quality of the query planner's estimates.

## The `default_statistics_target` Parameter

The `default_statistics_target` parameter:

- Is used to increase sampling for statistics collected for ALL columns
- Can improve query planner estimates
- Is set to 25 by default
- Can increase the time for `ANALYZE`, but can improve query planner's estimate
- Is overridden by `SET STATISTICS` on a column

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

115

For large tables, rather than examining every row to allow large tables to be analyzed in a small amount of time, `ANALYZE` takes a random sample of the table contents.

To increase sampling for all table columns, adjust the `default_statistics_target` configuration parameter.

Keep in mind that the `default_statistics_target` server configuration parameter applies to all columns by default. The default target value is 25.

A larger target value will increase the time needed to do `ANALYZE`, but may improve the quality of the query planner's estimates. especially for columns with irregular data patterns, may allow for more accurate query plans.

## Effect of Updating the Statistics Value



faadata	factontimeperformance	originairportid	0	4	293.083	{ATL,ORD,DFW,DEN,LAX,PHX,IAH,DTW,LAS,SFO,SLC,MCO,MSP,EWR,CLT,BOS,JFK,LGA,SEA,BWI,PHL,SAN,DCA,MDW,MEM}	{0.061137, 0.050212, 0.0408238, 0.0352377, 0.031034, 0.0276767, 0.0276485, 0.0242347, 0.0229087, 0.0208492, 0.020728, 0.0203696, 0.0193539, 0.0193257, 0.0187615, 0.0184793, 0.0181408, 0.0158838, 0.0157991, 0.0152631, 0.0140499, 0.01326, 0.0126675, 0.0121033, 0.0113133}   {ABE, ATL, AUS, BOS, CLE, CVG, DEN, DFW, DTW, EYW, HOU, IAH, JFK, LAX, LGA, MCO, MKE, OAK, ORD, PDX, PHX, RNO, SEA, SJC, SNA, YUM}
			(1 row)				

```
faa=# alter table factontimeperformance alter originairportid set statistics 50; analyze factontimeperformance;
ALTER TABLE
ANALYZE
```

 Sampling size has increased, improving statistics for the column

faadata	factontimeperformance	originairportid	0	4	302.021	{ATL,ORD,DFW,DEN,LAX,PHX,IAH,DTW,LAS,SFO,SLC,MCO,MSP,EWR,CLT,BOS,JFK,LGA,BWI,PHL,MDW,SAN,DCA,IAD,TPA,FL,CVG,SEA,CLT,BNA,HNL,HOU,ROU,PDX,MCI,SJC,OAK,SMF,DAL,AUS,SNA,SAT,MKE,PIT,IND,MSY,ABQ}	{0.0650631, 0.049255, 0.0410246, 0.0357173, 0.0311764, 0.0282248, 0.0269618, 0.0241095, 0.022875, 0.0212431, 0.0202065, 0.0194125, 0.0193841, 0.0190578, 0.0178516, 0.016986, 0.0167305, 0.0160778, 0.0160494, 0.0151128, 0.0148209, 0.0135377, 0.012743, 0.0117639, 0.0117213, 0.0113382, 0.0112104, 0.00987654, 0.00973464, 0.00939407, 0.00918121, 0.00869874, 0.00864198, 0.00861359, 0.00810274, 0.00800341, 0.00779055, 0.00776217, 0.00770541, 0.00767703, 0.00716617, 0.0071236, 0.00672627, 0.0066695, 0.0065276, 0.0064992, 0.00617284, 0.00611608, 0.00605932, 0.00573294}   {ABE, ATL, BNA, BOS, BWI, CIC, CLT, CVG, DCA, DEN, DFW, DTW, EWR, FAR, GNV, HOU, IAD, IAH, ITO, JFK, LAS, LAX, LGA, MCI, MDT, MEM, MKE, MSP, OAK, ONT, ORD, PDX, PHL, PHX, PSP, RNO, SAN, SEA, SFO, SJC, SLC, SNA, TPA, YUM}
			(1 row)				

Pivotal.

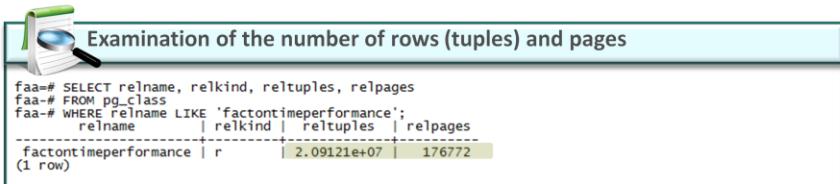
© 2015 Pivotal Software, Inc. All rights reserved.

116

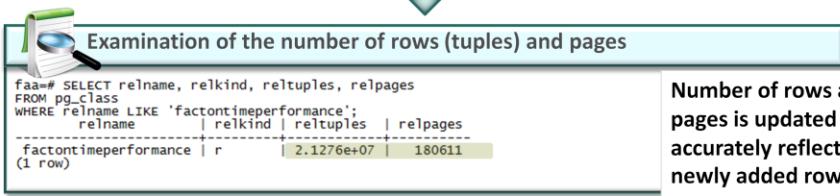
In this example, the statistics for the `originairportid` column of the `factontimeperformance` table is displayed. The sampling is based on the `statistics` value of 25, so the database sampled enough rows to obtain 25 distinct values in the `most_common_vals` column.

Next, the statistics on the `originairportid` table is increased to 50 and the table analyzed. This requests that the analyzer increase the sampling size so that there are now up to 50 distinct values in the `most_common_vals` column. By increasing the sampling size, the query planner will have a better insight into the makeup of the values within the column. This type of action can be performed on columns that are used more often in `WHERE` clauses. While it increases the amount of time required for analyzing the table, it gives a more accurate description of the makeup of the column, potentially improving the performance for the query.

## Updating the Total Number of Entries in a Table



```
faa=# analyze factontimeperformance;
ANALYZE
```



Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

117

The number of rows and pages, or disk blocks, used by a table are stored in the pg\_class table. This information is a component of statistics, used by the planner to build the query plan. The reltuples and relpages are not constantly updated and relies on the analysis performed by commands such as VACUUM, ANALYZE, and CREATE INDEX to update the values. As these values may be out of date, it could impact the final query plan generated for the table.

In this example, the number of rows is displayed as of the last ANALYZE. Separately, several thousand rows were added to the table. If the ANALYZE command is not executed, the pg\_class table continues to display the now outdated number of rows in the table.

Executing the ANALYZE command updates the number of rows as well as the number of disk blocks occupied by those rows.

## The `gp_analyze_relative_error` Parameter

The `gp_analyze_relative_error` server configuration parameter:

- Sets the estimated acceptable error in the cardinality of the table; a value of 0.5 is equivalent to an acceptable error of 50%
- Defaults to 0.25
- Allows the number of rows sampled to be increased if the relative error fraction is decreased
- Is set in the following manner:  
`gp_analyze_relative_error = .25`

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

118

The `gp_analyze_relative_error` server configuration parameter sets the estimated acceptable error in the cardinality of the table. A value of 0.5 is equivalent to an acceptable error of 50%. The default value is 0.25.

If the statistics collected are not producing good estimates of cardinality for a particular table attribute, decreasing the relative error fraction, or accepting less error, increases the number of rows sampled to determine the number of distinct non-null data values in a column. The number of distinct non-null data values in a column are stored in the `stadistinct` column of the `pg_statistic` table.

A larger target value will increase the time needed to do ANALYZE, but may improve the quality of the query planner's estimates.

## ANALYZE Best Practices

Consider the following:

- For large data warehouse environments it may not be feasible to run ANALYZE on the entire database or on all columns in a table
- Run ANALYZE for
  - Columns used in a JOIN condition
  - Columns used in a WHERE clause
  - Columns used in a SORT clause
  - Columns used in a GROUP BY or HAVING clause

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

119

It is important to maintain accurate statistics for optimal query performance. In large data warehouse environments it may not be feasible to run ANALYZE on an entire database or table due to time constraints. In this scenario it is important to generate statistics on columns used in a:

- JOIN condition
- WHERE clause
- SORT clause
- GROUP BY or HAVING clause

ANALYZE requires only a read lock on the table, so may be run in parallel with other database activity though not recommended when performing loads, INSERT, UPDATE, DELETE, and CREATE INDEX operations. Run ANALYZE after load, INSERT, UPDATE, DELETE, and CREATE INDEX operations.

## Lab: Improve Performance with Statistics

In this lab, you gather statistics on your data and analyze their behavior.

You will:

- Gather statistics and analyze queries

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

120

In this lab, you gather statistics on your data and analyze their behavior.

## Module 8: Performance Analysis and Tuning

### Lesson 5: Summary

During this lesson the following topics were covered:

- Collecting statistics about the database
- Improving performance by using `ANALYZE`
- Increasing sampling and statistics
- Identifying when it is best to use `ANALYZE`

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

121

This lesson discussed statistics and the need to provide updated statistics to the optimizer to more accurately predict the costs and time required to complete a query. The EXPLAIN ANALYZE and ANALYZE commands allow you to update metadata about tables, rows, and columns to improve the query plan that is generated by the optimizer. In addition to updating information on relations, increasing sampling size and statistics helps to fine tune the statistics collected on relations, which in turn can improve the query plan the optimizer generates.

# Module 8: Performance Analysis and Tuning

## Lesson 6: Indexing Strategies

In this lesson, you identify the index types available with Greenplum and decide when to use the index.

Upon completion of this lesson, you should be able to:

- List the supported index types for Greenplum
- Identify when to use an index
- Identify the costs associated with using indexes

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

122

If and when to index are key considerations when tuning your queries. You should be aware of when to use indexes and what types should be used, if you intend to implement them in the Greenplum Database.

In this lesson, you will:

- List the supported index types for Greenplum
- Identify when to use an index
- Identify the costs associated with using indexes

# Indexes

Most data warehouse environment operate on large volumes of data:

- With low selectivity
- Where sequential scan is the preferred method to read the data in a Greenplum MPP environment

For queries with high selectivity:

- Indexes may improve performance
- Avoid:
  - Indexes on frequently updated columns
  - Overlapping indexes
- Use bitmap indexes for columns with low cardinality
- Drop indexes before data load and recreate indexes after load
- Analyze after recreating indexes

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

123

In most data warehouse environment, queries operate on large volumes of data. In the Greenplum MPP environment a sequential is an efficient method to read data as each segment instance contains a portion of the data and all segment instances work in parallel. In an MPP shared nothing environment, segments are not competing for resources like in a traditional SMP environment.

For queries with high selectivity, indexes may improve query performance. First, run the query without indexes to determine if the query performance is acceptable. If it is determined that indexes are needed, consider best practices when creating and managing indexes.

Avoid indexes on columns that are frequently update and avoid overlapping indexes. Use bitmap indexes instead of a B-tree index for columns with very low cardinality. Low cardinality is a column that contains few values, such as `active` and `inactive`.

Drop indexes before loading data into a table. After the load re-create the indexes for the table. This will run an order of magnitude faster than loading data into a table with indexes.

## Using B-Tree or Bitmap Indexes

### B-Tree:

- Is used for fairly unique columns
- Is used for those columns that are single row queries
- Can be expensive

### Bitmap:

- Is used for low cardinality columns
- Is used when column is often included in predicate
- Is typically a fraction of the size of the indexed data
- Is best when data is queried instead of updated often

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

124

Bitmap indexes are one of the most promising strategies for indexing high dimensional data in data warehousing applications and decision support systems. These types of applications typically have large amounts of data and ad hoc queries, but a low number of DML transactions.

Bitmap indexes perform best for columns in which the ratio of the number of distinct values to the number of all rows, or degree of cardinality, in the table is small. A marital status column, which has only a few distinct values (single, married, divorced, or widowed), is a good choice for a bitmap index, and is considered low cardinality.

A bitmap index offers the best choice when less than 1% of the row count is distinct. Low cardinality with a high row count favors bitmap indexes.

Data with low cardinality and low row count should consider using a B-tree index.

In the end, the strategy you use depends on the nature of your data.

## Create Index Syntax

The following is the syntax to create an index:

```
CREATE [UNIQUE] INDEX [CONCURRENTLY]
    name ON table
    [USING method]      ( {column | (expression)}
    [opclass] [, ...] )
    [ WITH ( FILLFACTOR = value )
    [WHERE predicate]
```

The following is an example of how to create a bitmap index:

```
CREATE INDEX gender_bmp_idx ON employee USING bitmap
(gender);
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

125

Partial indexes are used as filters. These are considered indexes with a WHERE clause.

You can create an index on a function name or other expression. The expression has to be in the predicate so that the expression can use the index.

## B-Tree Index

The B-tree index:

- Supports single value row lookups
- Can be unique or non-unique; unique is supported only on a column that is, or is part of, the distribution key
- Can be single or multi-column
- Requires all columns in the index included in the predicate for the index to be used, if it is a multi-column index

```
CREATE INDEX transid_btridx  
  ON facts.transaction  
  USING BTREE (transactionid);
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

126

A B-tree index supports single value row lookups. It can be used on unique and non-unique values and with single or multi-column data.

You do not have to specify USING BTREE as it is the default type of index created when a terminology is not specified.

## Bitmap Index

A bitmap index:

- Can be a single column
- Is efficient for queries with multiple conditions on the predicate
- Provides very fast retrieval
- Is best for low cardinality columns, such as:
  - Gender
  - State / Province Code

The following are examples of bitmap indexes:

```
CREATE INDEX store_pharm_bmidx ON dimensions.store  
    USING BITMAP (pharmacy);  
CREATE INDEX store_grocery_bmidx ON dimensions.store  
    USING BITMAP (grocery);  
CREATE INDEX store_deli_bmidx ON dimensions.store  
    USING BITMAP (deli);
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

127

It is not recommended to have multi-column indexes with bitmap indexes. However, a bitmap index is most effective for queries that contain multiple conditions in the WHERE clause. Rows that satisfy some, but not all, conditions are filtered out before the table itself is accessed. This improves response time, often dramatically.

## Index on Expressions

Indexing on expressions:

- Should only be used when the expression appears often in query predicates
- Has a very high overhead maintaining the index during insert and update operations

The following shows how it is used:

```
CREATE INDEX lcase_storename_idx  
    ON store (LOWER(storename));
```

This syntax supports the following query:

```
SELECT * FROM store WHERE LOWER(storename) = 'top foods';
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

128

While an index can be used as part of an expression, care should be taken as it has a very high overhead when maintaining the index during an insert or update operation. This method should only be used when the expression appears often in query predicates, the WHERE clause.

In this example, the LOWER (storename) expression may be used often in queries, so you could create an index on that expression.

## Index with Predicate (Partial Index)

A partial index:

- Pre-selects rows based on predicate
- Is used to select small numbers of rows from large tables

The following is an example of a partial index:

```
CREATE INDEX canada_stores_idx  
    ON facts.transaction  
    WHERE storeid IN(8,32);
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

129

A partial index is created when the WHERE clause is presented as part of the CREATE INDEX statement. A partial index is an index that contains entries for only a portion of a table, usually a portion that is more useful for indexing than the rest of the table. For example, if you have a table that contains both billed and unbilled orders where the unbilled orders take up a small fraction of the total table and yet is most often selected, you can improve performance by creating an index on just that portion.

## Greenplum Indexes – Partitioned Tables

Consider only the most recent data should be considered for indexing, as in this example:

- A partitioned transaction table requires a B-tree index on the transaction id to support single row queries.
- Customer Support only needs to access the past 30 days of data.
- The transaction table has weekly partitions.

The solution is to index the 4 most recent partitions on a *rolling* basis.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

130

Often, only the most recent data should be considered for indexing based on the performance requirements of the user community.

For Example:

- A partitioned transaction table requires a B-tree index on the transaction id to support single row queries.
- Customer Support only needs to access the past 30 days of data.
- The transaction table has weekly partitions.

The solution for this problem is to Index the 4 most recent partitions on a rolling basis.

## To Index or Not to Index

Consider the following:

- Will the optimizer use the index?
- Is the column(s) used in query predicates?
  - Does the frequency of use justify the overhead?
  - Is the space available?
- Are you working with compressed append-only tables?



**Note:** Greenplum Database will automatically create PRIMARY KEY indexes for tables with primary keys.

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

131

There are several things to consider when deciding whether or not to use indexes:

- Will the optimizer use the index? If the optimizer does not use the index, you may be adversely affecting overall performance by maintaining an index that is not in use. Verify that the optimizer uses the index by issuing queries on production volumes. Additionally, consider that indexes can potentially lead to performance improvements for OLTP type workloads as opposed to BI query workloads, where OLTP transactions return a much smaller data set.
- Is the column(s) used in query predicates? If the column is constantly used in a query predicate, then there may be benefits to creating the index as it may decrease the number of rows selected from a large table.  
However, consider how much it costs to store the index, especially if it is a B-tree index. Sometimes, the index can be several times larger than the data stored in the table. Consider also how often the index is used. If it is used rarely, the costs may outweigh the benefits.
- Are you working with compressed append-only tables? If so, you can see some benefits with implementing queries where an index access method means only necessary rows are uncompressed.

## Index Costs

Indexes:

- Are not free
- Occupy space
- Incur overhead during inserts and updates
- Incur processing overhead during creation

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

132

Indexes are not free. They do require space. The worst single performance killer for queries are indexes. If needed, drop them first to reduce overhead. They require maintenance whenever the table is updated. Make sure that the indexes you create are actually being used by your query workload to justify creation and maintenance costs.

## Index Best Practices

Consider the following:

- Only create indexes when full table scans do not perform well
- Consider B-tree indexes for:
  - Single column lookups
  - Columns specified in the distribution key
- Consider bitmap indexes for commonly selected columns with low cardinality columns
- Use partial Indexes for queries that frequently access small subsets of much larger data sets
- Single level table partitions with bitmap and/or B-tree Indexes can be more efficient than multi-level partitions
- Clustered indexes can reduce disk seek time

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

133

Consider the following best practices when working with indexes:

- Verify that you do need an index. As a general rule, do not create indexes if you do not absolutely need them. If you find that table scans are slow and the data can support, try inserting an index. If you do not see any performance gains however, remove the index. Verify that for any index you create, it is used by the optimizer.
- If you find you have a need for a single column lookup, use a B-tree index for a single row lookup.
- Consider using bitmap indexes for commonly selected columns with low cardinality.
- Partial indexes can reduce the number of rows selected. This is particularly useful when working with large tables.
- Using an index for your data may be more efficient than using multi-level partitioning. Consider partitions created on gender or other low-cardinality values. An index can be easier to implement and may provide performance gains.
- Clustered indexes potentially reduce seek times by physically ordering records together according to the index. Randomly distributed records requires that Greenplum seeks across the disk for the records. However, moving them closer together can reduce the seek time on disk making the fetches more sequential. A good example of a clustered index is on a date range. Use the CLUSTER command to create a clustered index.

## Maintaining Indexes

To maintain overall performance:

- Check the disk space usage for your index
- Update or reindex your indexes if queries are spending too much time

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

134

The `gp_toolkit` administrative schema contains a number of views for checking index sizes. To see the total size of all indexes on a table, use the `gp_size_of_all_table_indexes` view. To see the size of a particular index, use the `gp_size_of_index` view. The index sizing views list tables and indexes by object ID and not by name. To check the size of an index by name, you must look up the table name in the `pg_class` by cross-referencing the `relname` column.

For B-tree indexes, a freshly-constructed index is somewhat faster to access than one that has been updated many times. This is because logically adjacent pages are usually also physically adjacent in a newly built index. It might be worthwhile to reindex periodically to improve access speed. Also, if all but a few index keys on a page have been deleted, there will be wasted space on the index page. A reindex will reclaim that wasted space. In Greenplum Database it is often faster to drop an index, using the `DROP INDEX` command and then recreate it with the `CREATE INDEX` command than it is to use the `REINDEX` command.

Bitmap indexes are not updated when changes are made to the indexed column(s). If you have updated a table that has a bitmap index, you must drop and recreate the index for it to remain current.

## Lab: Indexing Strategies

In this lab, you determine the requirements for indexes and create them as needed.

You will:

- Create indexes to support queries

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

135

In this lab, you determine the requirements for indexes and create them as needed.

## Module 8: Performance Analysis and Tuning

### Lesson 6: Summary

During this lesson the following topics were covered:

- Listing the supported index types for Greenplum
- Identifying when to use an index
- Identifying the costs associated with using indexes

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

136

This lesson focused more closely on indexes, the negative and positive impacts of using supported index types in Greenplum, when to use indexes, and how to determine if the use of the indexes is having a positive or negative impact.

## Module 8: Summary

Key points covered in this module:

- Combined data from multiple tables using JOINs
- Used EXPLAIN and EXPLAIN ANALYZE to help the optimizer determine how to handle a submitted query
- Improved query performance by keeping statistics up to date and tuning the database for sampling size and error conditions
- Determined when best to use an index and which type of index to use

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

137

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

- Combine data from multiple tables using JOINs.
- Use EXPLAIN and EXPLAIN ANALYZE to help the optimizer determine how to handle a submitted query.
- Improve query performance by keeping statistics up to date and tuning the database for sampling size and error conditions.
- Determine when it is best to use an index and what type of index to use.

This slide is intentionally left blank.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

138

# Module 9: Developing Reports Using Advanced SQL

This module examines the use of advanced SQL concepts in creating reports.

Upon completion of this module, you should be able to:

- Define OLAP and list the OLAP grouping sets available in Greenplum
- Use functions to manipulate and return data to a caller
- Improve query performance by following a number of performance enhancement tips

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

1

This module examines advanced reporting techniques, including OLAP and functions. OLAP groups and windowing functions offers benefits not only for coding, but also to improve performance.

In this module, you will:

- Define OLAP and list the OLAP grouping sets available in Greenplum
- Use functions to manipulate and return data to a caller
- Improve query performance by following a number of performance enhancement tips

# Module 9: Developing Reports Using Advanced SQL

## Lesson 1: Advanced Reporting Using OLAP

In this lesson, you use online analytic processing to quickly retrieve answers to multi-dimensional queries.

Upon completion of this lesson, you should be able to:

- Use set operations to manipulate data sets
- Identify OLAP grouping extensions
- Describe OLAP windowing functions
- Describe the global window specifications

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

2

Online analytic process queries are used to answer multidimensional questions asked by users.

In this lesson, you will:

- Use set operations to manipulate data sets
- Identify OLAP grouping extensions
- Describe OLAP windowing functions
- Describe the global window specifications

## Set Operations

Greenplum supports the following set operations as part of a SELECT statement:

- **INTERSECT** – Returns rows that appear in all answer sets
- **EXCEPT/MINUS** – Returns rows from the first answer set and excludes those from the second
- **UNION** – Returns a combination of rows from multiple SELECT statements with no repeating rows
- **UNION ALL** – Returns a combination of rows from multiple SELECT statements with repeating rows

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

3

Set operators:

- Manipulate the results sets of two or more queries by combining the results of individual queries into a single results set.
- Do not perform row level filtering.

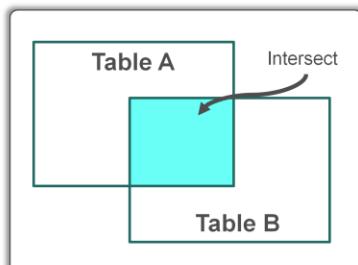
Set operations supported by Greenplum are:

- **INTERSECT** which returns rows that appear in all answer sets generated by individual SELECT statements.
- **EXCEPT** returns all rows from the first SELECT except for those which also selected by the second SELECT. This operation is the same as the MINUS operation.
- **UNION** combines the results of two or more SELECT statements. There will be no repeating rows.
- **UNION ALL** combines all the results of two or more SELECT statements. There may be repeating rows.

## Set Operations – INTERSECT

INTERSECT:

- Returns only the rows that appear in both SQL queries
- Removes duplicate rows



```
SELECT t.transid  
      c.custname  
  FROM facts.transaction t  
 JOIN dimensions.customer c  
    ON c.customerid = t.customerid
```

INTERSECT

```
SELECT t.transid  
      c.custname  
  FROM facts.transaction t  
 JOIN dimensions.customer c  
    ON c.customerid = t.customerid  
 WHERE t.transdate BETWEEN  
   '2008-01-01' AND '2008-01-21'
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

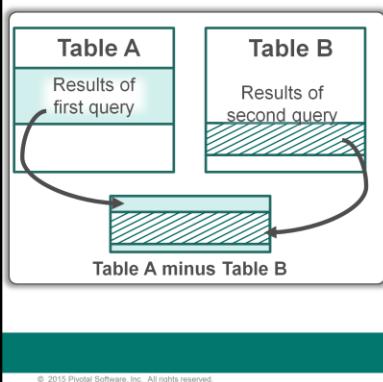
4

A set operation takes the results of two queries and returns only the results that appear in both result sets. Duplicate rows are removed from the final set returned.

## Set Operations – EXCEPT

EXCEPT:

- Returns all rows from the first SELECT statement
- Omits all rows that appear in the second SELECT statement



```
SELECT      t.transid  
           c.custname  
FROM        facts.transaction t  
JOIN        dimensions.customer c  
ON          c.customerid = t.customerid
```

EXCEPT

```
SELECT      t.transid  
           c.custname  
FROM        facts.transaction t  
JOIN        dimensions.customer c  
ON          c.customerid = t.customerid  
WHERE      t.transdate BETWEEN  
           '2008-01-01' AND '2008-01-21'
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

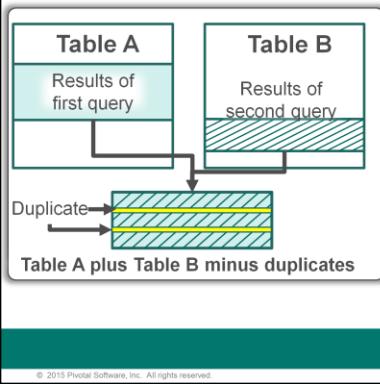
5

The EXCEPT set operation takes the distinct rows of the first query and returns all of the rows that do not appear in the result set of the second query.

## Set Operations – UNION

UNION:

- Combines rows from the first query with rows from the second query
- Removes duplicates or repeating rows



```
SELECT t.transid  
      c.custname  
  FROM facts.transaction t  
 JOIN dimensions.customer c  
    ON c.customerid = t.customerid  
 WHERE t.transdate BETWEEN  
      '2008-01-01' AND '2008-05-17'
```

UNION

```
SELECT t.transid  
      c.custname  
  FROM facts.transaction t  
 JOIN dimensions.customer c  
    ON c.customerid = t.customerid  
 WHERE t.transdate BETWEEN  
      '2008-01-01' AND '2008-01-21'
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

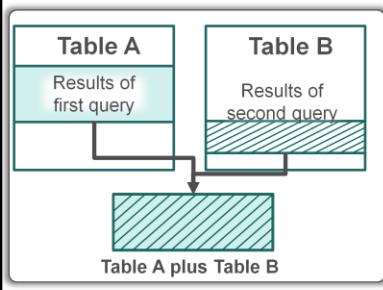
6

A union operation combines the results of the SELECT statement from the first table with the results from the query on the second table. The result set does not contain any repeating rows.

## Set Operations – UNION ALL

UNION ALL:

- Combines rows from the first query with rows from the second query
- Does not remove duplicate rows



```
SELECT t.transid  
      c.custname  
  FROM facts.transaction t  
 JOIN dimensions.customer c  
    ON c.customerid = t.customerid  
 WHERE t.transdate BETWEEN  
      '2008-01-01' AND '2008-05-17'
```

UNION ALL

```
SELECT t.transid  
      c.custname  
  FROM facts.transaction t  
 JOIN dimensions.customer c  
    ON c.customerid = t.customerid  
 WHERE t.transdate BETWEEN  
      '2008-01-01' AND '2008-01-21'
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

7

The UNION ALL set operation is like the UNION operation but it does not remove duplicate or repeating rows.

## What Is OLAP?

Online analytic processing:

- Is an approach to quickly provide answers to multi-dimensional queries
- Uses window functions that allow access to multiple rows in a single table scan
- Is enhanced with OLAP grouping extensions which are similar to GROUP BY but much more flexible

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

8

Part of broader category of BI, online analytic process is an approach to answer multi-dimensional queries. Databases configured for OLAP use a multidimensional data model that allows for complex analytical and ad-hoc queries.

Greenplum supports OLAP with window functions that provide access to multiple rows in a single table scan and grouping extensions, which provide flexibility when grouping result sets using the GROUP BY clause.

## Greenplum SQL OLAP Grouping Extensions

Greenplum supports the following grouping extensions:

- Standard GROUP BY
- ROLLUP
- GROUPING SETS
- CUBE
- `grouping(column [, ...])` function
- `group_id()` function

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

9

Greenplum introduced support for extensions to the standard GROUP BY clause, which is fully supported. These clauses can simplify the expression of complex groupings:

- **ROLLUP** – This extension provides hierarchical grouping.
- **CUBE** – Complete cross-tabular grouping, or all possible grouping combinations, is provided with this extension.
- **GROUPING SETS** – Generalized grouping is provided with the GROUPING SETS clause.
- **grouping function** – This clause helps identify super-aggregated rows from regular grouped rows.
- **group\_id function** – This clause is used to identify duplicate rows in grouped output.

## Standard GROUP BY Example

GROUP BY:

- Groups results together based on one or more columns specified
- Is used with aggregate statements

The following example summarizes product sales by vendor:

```
SELECT pn, vn, sum(prc*qty)
FROM sale
GROUP BY pn, vn
ORDER BY 1,2,3;
```

pn	vn	sum
100	20	0
100	40	2640000
200	10	0
200	40	0
300	30	0
400	50	0
500	30	120
600	30	60
700	40	1
800	40	1
(10 rows)		

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

10

The standard GROUP BY clause groups results together based on one or more columns specified. It is used in conjunction with aggregate statements, such as SUM, MIN, or MAX. This helps to make the resulting data set much more readable to a user.

The slide shows an example of a standard GROUP BY clause used to summarize product sales by vendor.

## Standard GROUP BY Example with UNION ALL

This example extends the previous example with the requirement that sub-totals and a grand total are added:

```
SELECT pn, vn, sum(prc*qty)
FROM sale
GROUP BY pn, vn
UNION ALL
SELECT pn, null, sum(prc*qty)
FROM sale
GROUP BY pn
UNION ALL
SELECT null, null,
sum(prc*qty)
FROM SALE
ORDER BY 1,2,3;
```

pn	vn	sum
100	20	0
100	40	2640000
100		2640000
200	10	0
200	40	0
200		0
300	30	0
300		0
400	50	0
400		0
500	30	120
500		120
600	30	60
600		60
700	40	1
700		1
800	40	1
800		1
		2640182

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

11

In this follow-on example, the requirements for the query have been extended to include sub-totals and a grand total. You would need to use a UNION ALL to continue the grouping and provide for the additional requirements.

## ROLLUP Example

The following example meets the requirement where the sub-total and grand totals are to be included:

```
SELECT pn, vn, sum(prc*qty)
FROM sale
GROUP BY ROLLUP(pn, vn)
ORDER BY 1,2,3;
```

pn	vn	sum
100	20	0
100	40	2640000
100		2640000
200	10	0
200	40	0
200		0
300	30	0
300		0
400	50	0
400		0
500	30	120
500		120
600	30	60
600		60
700	40	1
700		1
800	40	1
800		1
		2640182

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

12

This slide meets the requirements provided in the previous slide, but uses the ROLLUP grouping extension. ROLLUP allows you to perform hierarchical grouping and helps to reduce the code.

## GROUPING SETS Example

The following examples shows how to achieve the same results with the GROUPING SETS clause:

```
SELECT pn, vn, sum(prc*qty)
FROM sale
GROUP BY GROUPING SETS
( (pn, vn), (pn), () )
ORDER BY 1,2,3;
```

Summarize product sales by vendor

Subtotals for each vendor

Grand total

pn	vn	sum
100	20	0
100	40	2640000
100		2640000
200	10	0
200	40	0
200		0
300	30	0
300		0
400	50	0
400		0
500	30	120
500		120
600	30	60
600		60
700	40	1
700		1
800	40	1
800		1
		2640182

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

13

The GROUPING SETS extension allows you to specify grouping sets. If you use the GROUPING SETS clause to meet the earlier requirements so that it produced the same output as ROLLUP, it would use the following groups:

- (pn, vn) – This grouping summarizes product sales by vendor.
- (pn) – This grouping provides subtotal sales for each vendor.
- () – This grouping provides the grand total for all sales for all vendors.

## CUBE Example

CUBE creates subtotals for all possible combinations of grouping columns.

The following example

```
SELECT pn, vn, sum(prc*qty)
FROM sale
GROUP BY CUBE(pn, vn)
ORDER BY 1,2,3;
```

is the same as

```
SELECT pn, vn, sum(prc*qty)
FROM sale
GROUP BY GROUPING SETS
    ( (pn, vn), (pn),
      (vn), () )
ORDER BY 1,2,3;
```

pn	vn	sum
100	20	0
100	40	2640000
100		2640000
200	10	0
200	40	0
200		0
300		0
400	50	0
400		0
500	30	120
500		120
600	30	60
600		60
700	40	1
700		1
800	40	1
800		1
	10	0
	20	0
	30	180
	40	2640002
	50	0
		2640182

(24 rows)

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

14

A CUBE grouping creates subtotals for all of the possible combinations of the given list of grouping columns, or expressions. In terms of multidimensional analysis, CUBE generates all the subtotals that could be calculated for a data cube with the specified dimensions.

In the example shown on the slide, the additional grouping set of (vn) - subtotaling the sales by vendor, is included as part of the cube.

Note that  $n$  elements of a CUBE translate to  $2n$  grouping sets. Consider using CUBE in any situation requiring cross-tabular reports. CUBE is typically most suitable in queries that use columns from multiple dimensions rather than columns representing different levels of a single dimension. For instance, a commonly requested cross-tabulation might need subtotals for all the combinations of month, state, and product.

## GROUPING Function Example

Grouping distinguishes NULL from summary markers.

store	customer	product	price
s2	c1	p1	90
s2	c1	p2	50
s2		p1	44
s1	c2	p2	70
s1	c3	p1	40

(5 rows)

```
SELECT
store, customer, product,
sum(price),
    grouping(customer)
FROM dsales_null
GROUP BY
ROLLUP(store, customer,
       product);
```

```
SELECT * FROM dsales_null;
```

store	customer	product	sum	grouping
s1	c2	p2	70	0
s1	c2		70	0
s1	c3	p1	40	0
s1	c3		40	0
s1			110	1
s2	c1	p1	90	0
s2	c1	p2	50	0
s2	c1		140	0
s2		p1	44	0
s2			44	0
s2			184	1
			294	1

(12 rows)

© 2015 Pivotal Software, Inc. All rights reserved.

Pivotal.

15

When you use grouping extensions to calculate summary rows, such as sub-totals and grand totals, the output can become confusing if the data in a grouping column contains NULL values. It is hard to tell if a row is supposed to be a subtotal row or a regular row containing a NULL.

In the example shown on the slide, one of the rows shown where the customer field is NULL. Without the grouping id, you could misinterpret the sum of 44 as a subtotal row for store 2.

The GROUPING function returns a result for each output row, where:

- 1 represents a summary row.
- 0 represents grouped rows.

## GROUP\_ID Function

GROUP\_ID:

- Returns 0 for each output row in a unique grouping set
- Assigns a serial number >0 to each duplicate grouping set found
- Is useful when combining grouping extension clauses
- Can be used to filter output rows of duplicate grouping sets, such as in the following example:

```
SELECT a, b, c, sum(p*q), group_id()
FROM sales
GROUP BY ROLLUP(a,b), CUBE(b,c)
HAVING group_id()<1
ORDER BY a,b,c;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

16

In this example query, the combination of ROLLUP and CUBE produces:

- 11 grouping sets
- 7 DISTINCT grouping sets

The group\_id function can be used to filter out or identify duplicate grouping sets in the output.

GROUP BY ROLLUP (a,b), CUBE (b,c) is the same as:

- GROUP BY GROUPING SETS ( (a,b), (a), () ), GROUPING SETS ( (b,c), (b), (c), () )
- GROUP BY GROUPING SETS ((a,b,b,c), (a,b,b), (a,b,c), (a,b), (a,b,c), (a,b), (a,c), (a), (b,c), (b), ())

Where there are 11 total grouping sets but only 7 distinct grouping sets, where the groups are:

- (a,b,b,c) = (a,b,c) = (a,b,c)
- (a,b,b) = (a,b) = (a,b)
- (a,c)
- (b,c)
- (a)
- (b)
- ()

## OLAP Windowing Extensions

In this next section, you will examine the following topics on window functions:

- About window functions
- Constructing a window specification
- Using the `OVER` clause
- Using the `WINDOW` clause
- Using Built-in window functions

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

17

Greenplum supports the concept of OLAP analytical functions, also known as window functions. In this next section, you will examine:

- Key concepts of window functions and how they are used.
- The SQL syntax used to construct the window specification used by window functions:
- The use of the `OVER` clause, which is a window function classified by the use of the special `OVER` clause, used to define the window specification.
- The use of the `WINDOW` clause, which is a convenient feature for defining and naming window specifications that can be used in one or more window function `OVER` clauses of a query.
- The built-in window functions provided in Greenplum Database.

## About Window Functions

A window function:

- Is a class of function allowed only in the SELECT list
- Returns a value per row, unlike aggregate functions
- Has its results interpreted in terms of the current row and its corresponding window partition or frame
- Is characterized by the use of the OVER clause
- Defines the window partitions, or groups of rows to apply the function
- Defines ordering of data within a window
- Defines the positional or logical framing of a row in respect to its window

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

18

Window functions are a new class of functions introduced in Greenplum.

Window functions allow application developers to more easily compose complex OLAP queries using standard SQL commands. For example:

- Moving averages or sums can be calculated over various intervals.
- Aggregations and ranks can be reset as selected column values change.
- Complex ratios can be expressed in simple terms.

Window functions can only be used in the SELECT list, between the SELECT and FROM keywords of a query.

Unlike aggregate functions, which return a result value for each group of rows, window functions return a result value for every row, but that value is calculated with respect to the rows in a particular window partition (grouping) or window frame (row position within the window).

What classifies a function as a window function is the use of an OVER clause. The OVER clause defines the window of data to which the function will be applied. There are three characteristics of a window specification:

- Partitions (groupings) – a window function calculates the results for a row in respect to its partition.
- Ordering of rows within a window partition – some window function such as RANK require ordering.
- Framing – for ordered result sets, you can define a window frame that analyzes each row with respect to the rows directly above or below it.

## Defining Window Specifications (OVER Clause)

When defining the window function:

- Include an `OVER()` clause
- Specify the window of data to which the function applies
- Define:
  - Window partitions, using the `PARTITION BY` clause
  - Ordering within a window partition, using the `ORDER BY` clause
  - Framing within a window partition, using `ROWS` and `RANGE` clauses

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

19

All window functions must have an `OVER()` clause. The window function specifies the window of data to which the function applies.

It defines:

- Window partitions using the `PARTITION BY` clause.
- Ordering within a window partition using the `ORDER BY` clause).
- Framing within a window partition (`ROWS/RANGE` clauses).

## About the PARTITION BY Clause

The PARTITION BY clause:

- Can be used by all window functions
- Organizes result sets into groupings based on unique values
- Allows the function to be applied to each partition independently



**Note:** If the PARTITION BY clause is omitted, the entire result set is treated as a single window partition.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

20

The PARTITION BY clause:

- Can be used by all window functions. However, it is not a required clause. Windows that do not use the PARTITION BY clause present the entire result set as a single window partition.
- Organizes the result set into groupings based on the unique values of the specified expression or column.
- Allows the function to be applied to each partition independently.

## Window Partition Example

```
SELECT * ,  
row_number()  
OVER()  
FROM sale  
ORDER BY cn;
```

row_number	cn	vn	pn	dt	qty	prc
1	1	10	200	1401-03-01	1	0
2	1	30	300	1401-05-02	1	0
3	1	50	400	1401-06-01	1	0
4	1	30	500	1401-06-01	12	5
5	1	20	100	1401-05-01	1	0
6	2	50	400	1401-06-01	1	0
7	2	40	100	1401-01-01	1100	2400
8	3	40	200	1401-04-01	1	0

(8 rows)

```
SELECT * ,  
row_number()  
OVER(PARTITION  
BY cn)  
FROM sale  
ORDER BY cn;
```

row_number	cn	vn	pn	dt	qty	prc
1	1	10	200	1401-03-01	1	0
2	1	30	300	1401-05-02	1	0
3	1	50	400	1401-06-01	1	0
4	1	30	500	1401-06-01	12	5
5	1	20	100	1401-05-01	1	0
1	2	50	400	1401-06-01	1	0
2	2	40	100	1401-01-01	1100	2400
1	3	40	200	1401-04-01	1	0

(8 rows)

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

21

The example on the slide uses the `row_number` window function. This function returns a row number for each unique row in the result set.

In the first example, the `OVER` clause does not have a `PARTITION BY`. The entire result set is treated as one window partition.

In the second example, the window is partitioned by the customer number. Note that the result of row number is calculated within each window partition.

## About the ORDER BY Clause

The ORDER BY clause:

- Can always be used by window functions
- Is required by some window functions such as RANK
- Specifies ordering within a window partition

The RANK built-in function:

- Calculates the rank of a row
- Gives rows with equal values for the specified criteria the same rank

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

22

The ORDER BY clause is used to order the resulting data set based on an expression or column. It is always allowed in windows functions and is required by some window functions, including RANK. The ORDER BY clause specifies ordering within a window partition.

The RANK function is a built-in function that calculates the rank of a row in an ordered group of values. Rows with equal values for the ranking criteria receive the same rank. The number of tied rows are added to the rank number to calculate the next rank value. In this case, ranks may not be consecutive numbers.

## Using the OVER (ORDER BY...) Clause

```
SELECT vn, sum(prc*qty)
FROM sale
GROUP BY vn
ORDER BY 2 DESC;
```

vn	sum
40	2640002
30	180
50	0
20	0
10	0
(5 rows)	

```
SELECT vn, sum(prc*qty), rank()
OVER (ORDER BY sum(prc*qty)
DESC)
FROM sale
GROUP BY vn
ORDER BY 2 DESC;
```

vn	sum	rank
40	2640002	1
30	180	2
50	0	3
20	0	3
10	0	3
(5 rows)		

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

23

The slide shows an example of two queries that rank vendors by sales totals.

The first query shows a window function grouped on the vendor column, `vn`.

The second query uses the `RANK` function to output a ranking number for each row. Note that the `PARTITION BY` clause is not used in this query. The entire result is one window partition. Also, do not confuse `ORDER BY` of a window specification with the `ORDER BY` of a query.

## About Moving Windows

A moving window:

- Defines a set or rows in a window partition
- Allows you to define the first row and last row
- Uses the current row as the reference point
- Can be expressed in rows with the `ROWS` clause
- Can be expressed as a range with the `RANGE` clause

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

24

A moving or rolling window defines a set of rows within a window partition. When you define a window frame, the window function is computed with respect to the contents of this moving frame, rather than against the fixed contents of the entire window partition. Window frames can be row-based, represented by the `ROWS` clause, or value based, represented by a `RANGE`.

When the window frame is row-based, you define the number of rows offset from the current row. If the window frame is range-based, you define the bounds of the window frame in terms of data values offset from the value in the current row.

If you specify only a starting row for the window, the current row is used as the last row in the window.

## About Moving Windows (Cont)

A moving window:

- Is defined as part of a window with the ORDER BY clause as follows:

```
WINDOW window_name AS (window_specification)
where window_specification can be:
[window_name]
[PARTITION BY expression [, ...]]
[ORDER BY expression [ASC | DESC | USING operator] [, ...]
 [{RANGE | ROWS}
 { UNBOUNDED PRECEDING
 | expression PRECEDING
 | CURRENT ROW
 | BETWEEN window_frame_bound AND window_frame_bound }]]
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

25

## About Moving Windows (Continued)

## Designating the Moving Window

The window frame is defined with:

- UNBOUNDED | *expression* PRECEDING

... ROWS 5 PRECEDING

- UNBOUNDED | *expression* FOLLOWING

... ROWS 5 FOLLOWING

- BETWEEN *window\_frame* and *window\_frame*

... ROWS BETWEEN 5 PRECEDING AND UNBOUNDED FOLLOWING

- CURRENT ROW

... ROWS BETWEEN CURRENT ROW AND 5 FOLLOWING

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

26

The window frame can be defined as:

- **UNBOUNDED or *expression* PRECEDING** – This clause defines the first row of the window using the current row as a reference point. The starting row is expressed in terms of the number of rows preceding the current row. If you define the window frame for a ROWS window frame as 5 PRECEDING, the window frame starts at the fifth row preceding the current row. If the definition is for a RANGE window frame, the window starts with the first row whose ordering column value precedes that of the current row by 5. If the term UNBOUNDED is used, the first row of the partition acts as the first row of the window.
- **UNBOUNDED or *expression* FOLLOWING** – This clause defines the last rows of the window using the current row as a reference point. Similar to PRECEDING, the last row is expressed in terms of the number of rows following the current row. Either an expression or the term UNBOUNDED can be used to identify the last rows. If UNBOUNDED is used, the last row in the window is the last row in the partition.

## Designating the Moving Window (Continued)

- **BETWEEN *window\_frame\_bound* AND *window\_frame\_bound*** – This clause defines the first and last rows of the window, using the current row as a reference point. The first and last rows are expressed in terms of the number or rows preceding and following the current row, respectively. You can use other window frame syntax to define the bound. For example, BETWEEN 5 PRECEDING AND 5 FOLLOWING defines a window frame where the previous 5 rows and the next 5 rows from the current row are included in the moving window.
- **CURRENT ROW** – This clause references the current row in the partition. If a window frame is defined as BETWEEN CURRENT ROW AND 5 FOLLOWING, the window is defined starting with the current row and ending with the next 5 rows.

## Window Framing Example

A rolling window moves through a partition of data, one row at a time.

```
SELECT vn, dt,
       AVG(prc*qty)
    OVER (PARTITION BY vn
          ORDER BY dt
         ROWS BETWEEN
            2 PRECEDING AND
            2 FOLLOWING)
   FROM sale;
```

vn	dt	avg
10	03012008	30
20	05012008	20
30	05022008	0
30	06012008	60
30	06012008	60
30	06012008	60
40	06012008	140
40	06042008	90
40	06052008	120
40	06052008	100
50	06012008	30
50	06012008	10
(12 rows)		

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

28

While window framing clauses require an ORDER BY clause, not all window functions allow framing.

The ROWS and RANGE clauses specify a positional or logical rolling window that moves through a window partition of data.

In the example shown on the slide, the rolling frame applies to its partition, in this case, vendor, and ordering within that partition, date.

The example shows positional framing using the ROWS BETWEEN clause where the result is interpreted with respect to the CURRENT ROW position in the partition. The focus of the window frame moves from row to row within its partition only.

## Window Framing Example (Cont)

The diagram illustrates the movement of a window frame across four stages of data processing:

- Stage 1:** The first window frame (rows 0-3) has its last row (dt=06012008) highlighted with a dotted border.
- Stage 2:** The window frame has moved one row forward. The new last row (dt=06012008) is highlighted with a dotted border.
- Stage 3:** The window frame has moved another row forward. The new last row (dt=06012008) is highlighted with a dotted border.
- Stage 4:** The window frame has moved one more row forward. The new last row (dt=06012008) is highlighted with a dotted border.

vn	dt	avg
30	05022008	0
30	06012008	60
30	06012008	60
30	06012008	60
...		

vn	dt	avg
30	05022008	0
30	06012008	60
30	06012008	60
30	06012008	60
...		

vn	dt	avg
30	05022008	0
30	06012008	60
30	06012008	60
30	06012008	60
...		

vn	dt	avg
30	05022008	0
30	06012008	60
30	06012008	60
30	06012008	60
...		

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

29

The focus of the frame moves from the first selectable row in the window partition using the criteria, 2 preceding and 2 following, for the rolling window.

## Built-in Window Functions

Built-in Function	Description
cume_dist()	Calculates the cumulative distribution of a value in a group of values. Rows with equal values always evaluate to the same cumulative distribution value.
dense_rank()	Computes the rank of a row in an ordered group of rows without skipping rank values. Rows with equal values are given the same rank value.
first_value(expr)	Returns the first value in an ordered set of values.
lag(expr [,offset] [,default])	Provides access to more than one row of the same table without doing a self join. Given a series of rows returned from a query and a position of the cursor, LAG provides access to a row at a given physical offset prior to that position. If offset is not specified, the default offset is 1. default sets the value that is returned if the offset goes beyond the scope of the window. If default is not specified, the default value is null.



**Note:** Any aggregate function used with the OVER clause can also be used as a window function.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

30

The slide shows built-in window functions supported within Greenplum. These built-in functions require an OVER clause.

For more detailed information on the functions, refer to the *Greenplum Database Administrator Guide*.

## Built-in Window Functions (Cont)

Built-in Function	Description
<code>last_value(expr)</code>	Returns the last value in an ordered set of values.
<code>lead(expr [,offset] [,default])</code>	Provides access to more than one row of the same table without doing a self join. Given a series of rows returned from a query and a position of the cursor, LAG provides access to a row at a given physical offset after that position. If offset is not specified, the default offset is 1. default sets the value that is returned if the offset goes beyond the scope of the window. If default is not specified, the default value is null.
<code>ntile(expr)</code>	Divides an ordered dataset into a number of buckets (as defined by expr) and assigns a bucket number to each row.
<code>percent_rank()</code>	Calculates the rank of a hypothetical row R minus 1, divided by 1 less than the number of rows being evaluated (within a window partition).
<code>row_number()</code>	Assigns a unique number to each row to which it is applied (either each row in a window partition or each row of the query).

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

31

## Built-in Window Functions (Continued)

## Global Window Specifications

The WINDOW clause:

- Is useful for defining multiple window function queries
- Defines and names a window specification
- Lets you reuse window specifications throughout the query

```
SELECT  
RANK() OVER (ORDER BY pn),  
SUM(prc*qty) OVER (ORDER BY pn),  
AVG(prc*qty) OVER (ORDER BY pn)  
FROM sale;
```

The w1 window is used to call the code ORDER BY pn

```
SELECT  
RANK() OVER (w1),  
SUM(prc*qty) OVER (w1),  
AVG(prc*qty) OVER (w1)  
FROM sale  
WINDOW w1 AS (ORDER BY pn);
```

Pivotal.

The WINDOW clause is a convenient feature that allows you to define and name a window specification once and then refer to it multiple times throughout the query.

In the example shown on the slide, several window functions are using the same window specification in the OVER clause. You can use the WINDOW clause to define the window specification once, and then refer to it by name. In this example, the window specification is called w1.

## Lab: Advanced Reporting Using OLAP

In this lab, you write SQL statements using various OLAP functions to support a set of reporting requirements provided.

You will:

- Create reporting queries using OLAP functions

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

33

In this lab, you write SQL statements using various OLAP functions to support a set of reporting requirements provided.

# Module 9: Developing Reports Using Advanced SQL

## Lesson 1: Summary

During this lesson the following topics were covered:

- Using set operations to manipulate data sets
- Identifying OLAP grouping extensions
- Describing OLAP windowing functions
- Describing global window specifications

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

34

This lesson covered common set operations, OLAP grouping, and OLAP windowing functions that are used to present and summarize data for reporting purposes. These operations can greatly reduce the number of lines of SQL code required to return the same results and improve overall performance of the queries.

# Module 9: Developing Reports Using Advanced SQL

## Lesson 2: PostgreSQL Functions

In this lesson, you use PostgreSQL functions to access the database.

Upon completion of this lesson, you should be able to:

- Create functions and procedures
- Use the functions and procedures to return values
- Implement loops
- Set traps for error handling

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

35

Functions allow you to define and name a block of code that you can call by name during your development effort. Functions can reduce development time and provide you with a means of returning a value to a caller.

In this lesson, you will:

- Create functions and procedures
- Use the functions and procedures to return values
- Implement loops
- Set traps for error handling

## Types of Functions

Greenplum supports several function types, including:

- Query language functions where the functions are written in SQL
- Procedural language functions where the functions are written in:
  - PL/pgSQL
  - PL/Tcl
  - Perl
- Internal functions
- C-language functions



**Note:** Greenplum supports PL/pgSQL, PL/Perl, and PL/Python out of the box. Other extensions can be added with the gppkg utility and registered with the `createlang` utility.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

36

Greenplum supports a variety of methods for developing functions, including:

- Query language support for functions developed in SQL.
- Procedural language support for functions written in languages such as PL/PGSQL, which is a subset of PL/SQL, PL/Tcl, Perl, Python, and R, a programming language for statistical computing and graphics.
- Internal functions
- C-language functions

On installing Greenplum, PL/pgSQL is installed. All other languages can be installed to work with Greenplum.

Languages can be added to Greenplum with the Greenplum Package Manager utility and registered with the language handler utility, `createlang`. Greenplum:

- Has installed packages for PL/pgSQL, PL/Perl, and PL/Python.
- Has already registered PL/pgSQL in all databases by default, so no further action is required to enable it.
- Includes a language handler for PL/R, but the package is not pre-installed.
- Includes a package for PL/Tcl, but it is not enabled by default.

This lesson focuses primarily on query language and procedural language support.

## Greenplum Package Management

Package must be installed on the server acting as the primary master server.



### Example: Installing PL/R with Greenplum Package Management Utility

```
[gpadmin@mdw packages]# gppkg --install plr-1.0-rhel5-x86_64.pkg
20120130:10:21:58:gppkg:mdw:gpadmin-[INFO]:-Starting gppkg with args: --install
plr-1.0-rhel5-x86_64.gppkg
20120130:10:21:59:gppkg:mdw:gpadmin-[INFO]:-Installing package plr-1.0-rhel5-
x86_64.gppkg
20120130:10:22:01:gppkg:mdw:gpadmin-[INFO]:-Transferring package to segment sdw1
20120130:10:22:01:gppkg:mdw:gpadmin-[INFO]:-Transferring package to segment sdw2
20120130:10:22:06:gppkg:mdw:gpadmin-[INFO]:-Transferring package to standby smdw
20120130:10:22:08:gppkg:mdw:gpadmin-[INFO]:-Validating rpm installation
cmdStr='rpm --test -i /usr/local/greenplum-db/.tmp/plr-1.0-1.x86_64.rpm
/usr/local/greenplum-db/.tmp/R-2.13.0-1.x86_64.rpm --dbpath
/usr/local/greenplum-db/share/packages/database --prefix /usr/local/greenplum-
db/.'
20120130:10:22:20:gppkg:mdw:gpadmin-[INFO]:-Please source your
$GPHOME/greenplum_path.sh file and restart the database.
You can enable PL/R by running createlang plr -d mydatabase.
20120130:10:22:30:gppkg:mdw:gpadmin-[INFO]:-plr-1.0-rhel5-x86_64.gppkg
successfully installed.
```

Greenplum provides packages for PostGIS, PL/Java, PL/R, PL/Perl, and Pgcrypto.

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

37

Before we discuss functions, let us examine the package management feature available starting with Greenplum Database 4.2.

The Greenplum Package Manager, gppkg, provides an interface for installing extensions, such as pl/R, pgcrypto, and PostGIS to name a few. You can install, update, or remove packages to and from the environment.

Greenplum provides the available packages from the EMC Download Center. It manages the installation and distribution of packages to all systems in the Greenplum cluster. In addition, it is automatically called to install the extensions during an upgrade or expansion of the Greenplum cluster. Some of these packages may require additional manual steps to make them available to the Greenplum Database. The list of supported extensions currently supported include:

- PostGIS
- PL/Java
- PL/R
- PL/Perl
- Pgcrypto

The Greenplum Package Management utility does not currently support user-custom modules.

## Query Language Function – Rules

### An SQL function:

- Executes an arbitrary set of SQL commands
- Returns the results of the last query in the list
- Returns the first row of the result set of the last statement executed
- Can return a set of a previously defined data type
- Must end with a semicolon at the end of each statement in the body of the function

### SQL functions:

- May contain SELECT statements
- May contain DML statements, such as INSERT, UPDATE, or DELETE statements
- May not contain ROLLBACK, SAVEPOINT, BEGIN, or COMMIT commands
- Must use SELECT in the last line unless the return type is void.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

38

The query language function is a list of SQL statements that are wrapped in a function so that it is easier for you to call it when needed. An SQL function:

- Executes an arbitrary set of SQL statements within the body of the function.
- Returns the results of the last query in the list.
- Returns the first row of the result set of the last statement executed in the function body.
- Can return a set of previously defined data types.
- Must end with a semicolon at the end of each SQL statement.

You:

- Can include SELECT statements.
- Cannot include rollback, savepoint, begin, or commit commands.
- Support these end-user functions yourself.

One of the main strengths of a function is to return a value. When defining the function, the last statement must select something. It does not return a void by default.

## Query Language Function Structure

When developing SQL functions:

- Use doubled single quote syntax to reference quoted values in the query
- You can pass in one or more parameters may be passed to a function. Parameters are:
  - A sequential list referenced by \$n notation
  - Start at \$1 for the first parameter and increase for each additional parameter

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

39

The query language requires that quoted values use the doubled single-quote syntax, where two single quotes are used.

If you are passing parameters into a function, the parameters will be referenced by \$n, where n is a sequential list of numbers, starting with 1. The first parameter is referenced as \$1, while the second is referenced as \$2. Each additional parameter is assigned the next sequential number in the series.

## Creating, Modifying, and Dropping Functions

Action	SQL Syntax
Create a function	CREATE FUNCTION
Replace a function	CREATE OR REPLACE FUNCTION
Change a function	ALTER FUNCTION
Drop or remove a function	DROP FUNCTION

There are several things to note when managing functions:

- Functions operating on tables must be created in the same schema as the table
- Removing a statement requires that the `DROP` statement include the parameter types. For example:  
`DROP FUNCTION DIMENSIONS.getcust(int);`

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

40

Functions that operate on tables must be created in the same schema. If you modify a table, you must have access to a schema. You:

- Create a function with the `CREATE FUNCTION` command. You must have `CREATE` access to the schema to create a function. A function can be created with or without parameters.
- Replace an existing function with the `CREATE OR REPLACE FUNCTION` command. This command either creates a function if one did not exist before, or replaces an existing function. If you are replacing an existing function, you must specify the same number of parameters and the same data types found in the original function. If not, you are actually creating a new function.
- Change a function with the `ALTER FUNCTION` command. You must own the function before you can modify it. If the function is to be created in another schema, you must have `CREATE` privilege on that schema.
- Drop or remove a function with the `DROP FUNCTION` command. Because you can have multiple functions with the same name but different number of parameters and/or parameter types, you must include the appropriate number of parameters and parameter types as part of the command. You must also be the owner of the function to remove the function from the schema.

## Query Language Function – Example

The following example shows a function that has no parameters or return set:

```
CREATE FUNCTION public.clean_customer()
  RETURNS void AS 'DELETE FROM dimensions.customer
  WHERE state = ''NA''; '
  LANGUAGE SQL;
```

The function is called as follows:

```
SELECT public.clean_customer();

clean_customer
-----
(1 row)
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

41

In this example, a function is created to remove invalid customers from the dimensions.customer table.

To use the function, issue a SELECT statement against the function. The function runs from the master, which evaluates and pushes all of the SQL in the statement down to the segments. The result is returned to the master.

## Query Language Function – With Parameter

The following example shows a function that has no parameters or return set:

```
CREATE FUNCTION public.clean_specific_customer
(which char(2))
RETURNS void AS 'DELETE FROM dimensions.customer
WHERE state = $1; '
LANGUAGE SQL;
```

The function is called as follows:

```
SELECT public.clean_specific_customer('NA');

clean_specific_customer
-----
(1 row)
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

42

In this example, a parameter is passed in. You can pass in a character as the data type. You do not need to specify a name for the parameter. The name `which` does not need to be in the code. The function definition will accept the data type, in this case, `char(2)`, without it being named, since it is not used anywhere else in the function.

## SQL Functions on Base Types

In this example, a single value is returned to the caller:

```
CREATE FUNCTION public.customer_cnt(char)
  RETURNS bigint AS 'SELECT COUNT(*)
    FROM dimensions.customer
    WHERE state = $1; '
  LANGUAGE SQL;
```

The function is called as follows:

```
SELECT public.customer_cnt('WA');

customer_cnt
-----
176
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

43

In this example, the char data type is being passed into the function, `public.customer_cnt`. Since it performing an aggregation with `COUNT (*)`, it evaluates on the master. The `FROM` predicate gets evaluated on the segments.

## SQL Functions on Composite Types

The following example shows how to pass a composite parameter, in this case, a table:

```
CREATE FUNCTION facts.viewnewtaxamt(transaction) RETURNS  
numeric AS $$ Start function body  
    SELECT $1.taxamt * .90; Function body  
$$ LANGUAGE SQL;  
End function body
```

The function is called as follows:

```
SELECT transid, transdate, taxamt,  
facts.viewnewtaxamt(transaction)  
    FROM transaction;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

44

Whenever you pass in a parameter, you can identify it as:

- A base or primitive type, such as `integer`, `char`, or `varchar`.
- A composite parameter which is a relation-based parameter.
- A domain parameter.

In this example, you pass the parameter `transaction`, which is relation-based. When you call this function, you select the various columns in the table. The value is multiplied by .9 and returned as numeric.

## SQL Functions with Output Parameters

The following function does not call the `RETURNS` clause, instead passing a value out of the function using an `OUT` parameter:

```
CREATE FUNCTION public.cust_cnt(IN whichstate char ,  
                                OUT MyCount bigint)  
AS 'SELECT COUNT(*)  
     FROM dimensions.customer  
    WHERE state = $1; '  
LANGUAGE SQL;
```

The function is called as follows:

```
SELECT public.cust_cnt('WA');  
customer_cnt  
-----  
176
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

45

This example does not have a return clause. Within the parameter block, you define variables as:

- `IN` to show which variables are being passed into the function.
- `OUT` to specify the variables that are passed out of the function. This is a shortcut for not having a return clause.

Specifically using the `RETURNS` clause or defining `IN` and `OUT` parameters is a matter of your coding preference and style.

## SQL Functions as Table Sources

The following shows a function that accepts an integer and returns a composite parameter:

```
CREATE FUNCTION dimensions.getcust(int)
    RETURNS customer AS $$ 
    SELECT *
        FROM dimensions.customer WHERE customerid = $1;
$$ LANGUAGE SQL;
```

The function is called as follows:

```
SELECT *, UPPER(custname)
    FROM dimensions.getcust(100);
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

46

A function can be used as a table source. This lets you apply a filter or security check, wrapped around a table. Whenever a user queries the table, the results are returned in the correct format.

In this example, the `dimensions.getcust (int)` function returns a result set where the customer is equal to customer ID. This makes it much easier for a user to issue a call to the table without needing to specify the syntax provided in the function.

## SQL Functions Returning Sets

The following example returns all rows to the caller using **SETOF** instead of returning only the first row:

```
CREATE FUNCTION dimensions.getstatecust(char)
    RETURNS SETOF customer AS $$%
        SELECT *
        FROM dimensions.customer WHERE state = $1;
$$ LANGUAGE SQL;
```

The function is called as follows:

```
SELECT *,UPPER(city)
FROM dimensions.getstatecust('WA');
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

47

When an SQL function is declared as returning **SETOF** *data\_type*, the final **SELECT** query in the function is executed to completion. Each row it outputs is returned as an element of the result set. If you do not specify a return as a **SETOF**, only the first row is returned.

Now that you have an understanding of the structure of a function, let us examine function overloading.

## Function Overloading

Function overloading:

- Lets you define multiple functions with the same name
- Must have different input parameter types or number
- Allows the appropriate version of the function to be called by the optimizer based on the input data type and number of arguments

```
CREATE FUNCTION get_customer_ids(int)
...
LANGUAGE SQL;
```



```
CREATE FUNCTION get_customer_ids(int)
...
LANGUAGE SQL;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

48

Function overloading allows you to build several functions with the same name, but with different types or number of parameters. For example, creating a function called `get_customer_ids (int)` is different from the function called `get_customer_ids (char)`. If you pass in a character value to the function, the second function is called. Two functions are considered the same if they have the same names and input argument types.

## Function Overload – Example

```
CREATE FUNCTION dimensions.getcust(int)
    RETURNS customer AS $$  
SELECT *  
FROM dimensions.customer WHERE customerid = $1;  
$$ LANGUAGE SQL;
```

This function accepts  
an INTEGER parameter

```
CREATE FUNCTION dimensions.getcust(char)
    RETURNS customer AS $$  
SELECT * FROM dimensions.customer WHERE state = $1;  
$$ LANGUAGE SQL;  
select *,upper(custname) from dimensions.getcust('WA') ;  
select *,upper(custname) from dimensions.getcust(1);
```

This function accepts a  
CHAR parameter

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

49

In this example, you have two functions defined with the same name. The first passes in an integer whereas the second passes in a character value. These functions have two different behaviors, but the same name. The optimizer will decide which function to call based on the parameters you pass in.

## Function Volatility Categories

Functions belong to one of the following categories:

- **IMMUTABLE** – Functions rely only on information passed in and always returns the same values given the same argument values
- **STABLE** – Functions:
  - Consistently return the same result for the same argument values in a single table scan
  - Can return different values across SQL statements
- **VOLATILE** – Functions in this category:
  - Are evaluated on each row
  - Can return different values within a single table scan

**Note:** IMMUTABLE and VOLATILE functions cannot be executed at the segment level.



Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

50

The volatility category gives the optimizer a general idea of what the function will do and whether it falls into one of the following areas:

- **IMMUTABLE** – In an immutable function, the value is not going to change. This function relies on information provided in the argument list and will always return the same values for the same argument values.
- **STABLE** – Each table scan means that the function is evaluated once and uses the updated value. Another table scan will yield different results. Functions whose parameters depend on database lookups or parameter variables are classified as STABLE. Functions in the `current_timestamp` family qualify as stable.
- **VOLATILE** – The function is evaluated on each row. In this case, the function value can change within a single table scan. Examples of volatile functions include `random()`, `currval()`, and `timeofday()`.

## Function Volatility – IMMUTABLE

An IMMUTABLE function:

- Cannot modify the database
- Is guaranteed to return the same results given the same arguments
- Allows the optimizer to pre-evaluate the function when a query calls it with constant arguments
- Can be executed at the segment level

Consider the following:

The query, `SELECT ... WHERE x = 2 + 2;` can be simplified with `SELECT ... WHERE x = 4;` because the function underlying the integer addition operator is marked IMMUTABLE.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

51

**IMMUTABLE** functions, considered the most stable of functions, cannot modify the database. All of your operators are considered immutable.

**IMMUTABLE** functions are guaranteed to return the same result given the same set of arguments every time. This allows the optimizer to pre-evaluate the function when a query calls it with the same arguments.

This type of function can be run at the segment level because the system can guarantee its behavior.

## Function Volatility – STABLE

A STABLE function:

- Cannot modify the database
- Is guaranteed to return the same results given the same arguments for all rows within a single statement
- May not return the same results across SQL statements
- Allows the optimizer to optimize multiple calls of the function to a single call
- Is safe to use as part of an expression in an index scan condition

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

52

As with IMMUTABLE functions, STABLE functions cannot modify the database. Their use is restricted and so cannot be used at the segment level.

A function defined as STABLE will return the same results given the same arguments in a single table scan. However, the results may differ across multiple SQL statements. This does allow the optimizer to optimize multiple calls of the function in a single call.

## Function Volatility – VOLATILE

A VOLATILE function:

- Can do anything, including modifying the database.
- Can return different results on successive calls with the same arguments.
- Tells the optimizer to make no assumptions about the behavior of the function
- Causes a query accessing the volatile function to re-evaluate the function at every row where its value is needed

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

53

The use of VOLATILE functions is restricted in Greenplum.

A VOLATILE function is capable of doing anything, including modifying the database. The value of the function changes on each successive call even with the same arguments in a single table scan. The optimizer cannot make any assumptions about the behavior of the function.

Now that you have examined building functions in SQL, we will examine building functions in the PL/pgSQL procedural language.

## Procedural Language Functions

PL/pgSQL procedural language for Greenplum:

- Can be used to create functions and procedures
- Adds control structures to the SQL language
- Can perform complex computations
- Inherits all user-defined types, functions, and operators
- Can be defined to be trusted by the server
- Is *relatively* easy to use

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

54

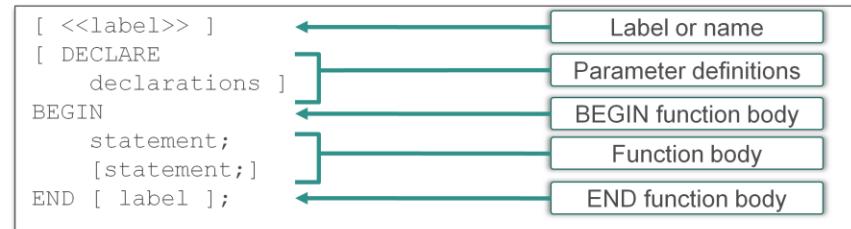
Procedural language support for Greenplum is in PL/pgSQL. This language is registered in Greenplum on installation and is available to all databases. It allows support for flow control, structures, and returns. This is an enhancement over programming with SQL statements allow where it:

- Adds control structures to the SQL language.
- Can perform complex computations.
- Inherits all user-defined types, functions, and operators.
- Is relatively easy to use.

## Structure of PL/pgSQL

A block:

- Holds the complete text of the function
- Is defined as:



Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

55

A function defined in PL/pgSQL must be defined within a block. The structure of the block:

- May contain a label.
- Must declare a parameter for each value passed in.
- Starts and ends with BEGIN and END respectively to define the boundaries of the block.
- Contains a number of statements within the BEGIN/END block. A block may also contain inner or subblocks, where you define and user local variables not visible to the outer block.

## Structure of PL/pgSQL – Example

```
CREATE FUNCTION somefunc() RETURNS integer AS $$  
DECLARE quantity integer := 30;  
BEGIN  
RAISE NOTICE 'Quantity here is %', quantity;  
-- Prints 30  
quantity := 50;  
-- Create a subblock  
DECLARE quantity integer := 80;  
BEGIN  
RAISE NOTICE 'Quantity here is %', quantity;  
-- Prints 80  
RAISE NOTICE 'Outer quantity here is %',  
outerblock.quantity;  
-- Prints 50  
END;  
RAISE NOTICE 'Quantity here is %', quantity;  
-- Prints 50  
RETURN quantity;  
END;  
$$ LANGUAGE plpgsql;
```

Inner BEGIN/END block

Outer BEGIN/END block

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

56

This example shows the structure of a PL/pgSQL function, where within the innermost BEGIN/END block, you:

- Print the value of quantity as it was declared right before the innermost BEGIN/END block.
- Print the value of the quantity variable as defined in the outer block.

The value of quantity:

- Was initially declared as 30 for the first BEGIN/END block.
- Updated directly to 50.
- Declared as 80 for the inner block.
- Is still 50 after the inner block has completed execution.

## PL/pgSQL Declarations

All variables used in a block:

- Must be declared in the declarations section of the block
- Is considered local to that block
- Are initialized to the declared value each time the block is called
- **The syntax is as follows:**

```
name [ CONSTANT ] type [ NOT NULL ] [ { DEFAULT | := }  
expression ];
```

The following are examples of variable declarations:

```
user_id integer;  
quantity numeric(5);  
url varchar;  
myrow tablename%ROWTYPE;  
myfield tablename.columnname%TYPE;  
somerow RECORD;
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

57

When you declare a block, you must define the name and data type for the block. Everything else is optional.

The variable is considered local to that block, meaning an outer variable is not aware of the definition.

## Defining Data Types in PL/pgSQL

The %TYPE:

- Acts as a placeholder for the data type of a variable or table column
- Copies the data type of the object being referenced
- Can be used to declare variables that will hold database values
- Is called in the following way:  
variable%TYPE

The following is an example of using %TYPE:

```
custid customer.customerid%TYPE  
tid transaction.transid%TYPE
```

Pivotal.

By using %TYPE as a place holder for the data type, you do not need to know the data type of the structure you are referencing. Most importantly, if the data type of the referenced item changes in the future, such as if you change the type of customerid from integer to real, you might not need to change your function definition.

%TYPE is particularly valuable in polymorphic functions, as the data types needed for internal variables can change from one call to the next. Appropriate variables can be created by applying %TYPE to the function arguments or result placeholders.

## PL/pgSQL Row Types

%ROWTYPE:

- Creates a row variable that can be declared to have the same type as the rows of an existing table or view
- Provides access to the user-defined columns of a table row, not the OID or other system columns
- Is called in the following way:  
table\_name%ROWTYPE

The following is an example of how to use %ROWTYPE:

```
cust customer%ROWTYPE  
trans transaction%ROWTYPE
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

59

A row type is a variable of a composite type. You can define a row type for rows of a specific table you are working with. This lets you hold an entire row for a SELECT statement, as long as the columns of the query match the declared type of the variable.

## PL/pgSQL Record Types

Record variables:

- Are similar to row-type variables
- Have no pre-defined structure
- Is not a true data-type
- Act as place holders for the row structure of the row they are assigned during a SELECT or FOR command
- Can have its substructure changed each time it is assigned
- Is defined in the following way:  
name RECORD;

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

60

The record structure changes based on what is assigned to it. This is ideal if you do not know the row structure you are working with. In this sense, it acts as a place holder for the variable.

## PL/pgSQL Basic Statements

The following are basic statements found in PL/pgSQL code:

- An assignment statement is defined as follows:  
`variable := expression;`
- Executing DML with no RETURN is as follows:  
`PERFORM myfunction(myparm1, myparm2);`
- Obtaining single row results and storing them into a variable is performed as follows:  
Use `SELECT ... INTO mytarget%TYPE;`

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

61

The result of a SQL command yielding a single row, possibly of multiple columns, can be assigned to a record variable, row-type variable, or list of scalar variables. This is done by writing the base SQL command and adding an `INTO` clause.

If a row or a variable list is used as target, the result columns of the query must exactly match the structure of the target with the appropriate number and data types of the column, or a run-time error occurs.

When a record variable is the target, it automatically configures itself to the row type of the query result columns.

The `INTO` clause can appear almost anywhere in the SQL command. Customarily, it is written either just before or just after the list of select expressions in a `SELECT` command, or at the end of the command for other command types.

## PL/pgSQL – Executing Dynamic SQL

Dynamic commands:

- Are permissible in PL/pgSQL functions
- Normally involve accessing different tables or different data types each time they are executed
- Are called as part of the EXECUTE statement:

```
EXECUTE command-string [ INTO [STRICT] target ];
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

62

Substitution of PL/pgSQL variables is not performed on the computed command string. Any required variable values must be inserted in the command string as it is constructed. Instead, the command is prepared each time the statement is run. The command string can therefore be dynamically created within the function to perform actions on different tables and columns.

The INTO clause has the following conditions:

- It specifies where the results of a SQL command returning rows should be assigned.
- If a row or variable list is provided, it must exactly match the structure of the query result set.
- When a record variable is used, it will configure itself to match the result structure automatically.
- If multiple rows are returned, only the first will be assigned to the INTO variable.
- If no rows are returned, NULL is assigned to the INTO variable(s).
- If no INTO clause is specified, the query results are discarded.

When working with dynamic commands you will often have to handle escaping of single quotes. The recommended method for quoting fixed text in your function body is dollar quoting.

## PL/pgSQL – Dynamic SQL Example

When working with dynamic values:

- Values may contain quote characters
- Safeguard your data by using:
  - `quote_ident` for expressions containing column and table identifiers
  - `quote_literal` for expressions with literal strings in the command

The following is an example of how these functions are used:

```
EXECUTE 'UPDATE tbl SET '
|| quote_ident(colname)
|| ' = '
|| quote_literal(newvalue)
|| ' WHERE key = '
|| quote_literal(keyvalue);
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

63

The example shown on the slide demonstrates the use of the `quote_ident` and `quote_literal` functions.

Dynamic values that are inserted into the constructed query require special handling since they may also contain quote characters. When working with dynamic values:

- Expressions containing column and table identifiers should be passed to `quote_ident`.
- Expressions containing values that should be literal strings in the constructed command should be passed to `quote_literal`.

Both take the appropriate steps to return the input text enclosed in double or single quotes respectively, with any embedded special characters properly escaped.

## PL/pgSQL – Getting the Results

To determine the effects of a command:

- Use the `GET DIAGNOSTICS` command, which is called as:  
`GET DIAGNOSTICS variable = item [ , ... ];`
- Check the special variable, `FOUND`, which:
  - Is of type `BOOLEAN`
  - Starts out false within each PL/pgSQL function call

The following command allows retrieval of system status indicators with the `GET DIAGNOSTICS` command:

```
GET DIAGNOSTICS integer_var = ROW_COUNT;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

64

There are several variables you can access from the system when performing a command. The `GET DIAGNOSTICS` command is used to obtain system status indicators after executing a command.

The `FOUND` variable, which is of type `BOOLEAN`, starts out as false for each block, but adjusts depending on the results of the calls made.

The full listing of variables you can retrieve are available on the PostgreSQL website.

## PL/pgSQL – FOUND

Statement	Behavior of FOUND
SELECT INTO	<ul style="list-style-type: none"><li>• TRUE if a row is assigned</li><li>• FALSE if the row is not assigned</li></ul>
PERFORM	<ul style="list-style-type: none"><li>• TRUE if one or more rows are produced</li><li>• FALSE if no rows are produced</li></ul>
UPDATE/INSERT/DELETE	<ul style="list-style-type: none"><li>• TRUE if one or more rows is affected</li><li>• FALSE if no rows are affected</li></ul>
FETCH	<ul style="list-style-type: none"><li>• TRUE if a row is returned</li><li>• FALSE if no rows are returned</li></ul>
MOVE	<ul style="list-style-type: none"><li>• TRUE if the cursor is repositioned</li><li>• FALSE if the cursor is not moved</li></ul>
FOR	<ul style="list-style-type: none"><li>• TRUE if the statement iterates one or more times</li><li>• FALSE if the statement does not</li></ul>

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

65

FOUND is a local variable within each PL/pgSQL function. Any changes to the variable affect only the current function. The variable is set by a different statements, including:

- SELECT INTO
- PERFORM
- UPDATE
- INSERT
- DELETE
- FETCH
- MOVE
- FOR

## PL/pgSQL – Control Structures

Control structures:

- Are probably the most useful and important part of PL/pgSQL
- Allow you to manipulate PostgreSQL data in flexible and powerful way
- Use flow control statements such as:
  - RETURN
  - Conditional statements
  - Simple loops
  - Looping through query results
  - Trapping errors

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

66

Control structures allows you to manipulate your data with flow control statements, such as:

- RETURN statements
- Conditional statements that include IF/THEN/ELSE blocks
- Simple loops, such as LOOP, CONTINUE, and WHILE
- Looping through query results FOR...IN...LOOP statements
- Trapping errors

## PL/pgSQL – Returning from a Function

When returning values to a caller:

- RETURN:
  - With an expression terminates the function and returns the value of expression to the caller
  - Lets you return a void and exit the function early
  - Can be called with no expression to return the current values of the output parameter variables
- RETURN NEXT:
  - Works on the next row after returning a value
  - Does not actually return from the function
  - Is specified when the function returns SETOF *data\_type*
  - Will require a RETURN statement to return from the function
- The return value of a function cannot be left undefined

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

67

RETURN and RETURN NEXT are two flow control commands that allow you to return data from a function.

RETURN:

- Returns a value and terminates the function.
- Lets you return a void and exit the function early if you declared the function to return a void. Do not write a statement after the RETURN statement however as it does not get evaluated.
- Can be called with no expression when the function defines output parameter variables. The function will still return the values of the output parameter variables.

RETURN NEXT:

- Works on the next row after returning a value.
- Does not return from the function, instead saving the value of the expression on that iteration.
- Is used when the function returns SETOF *data\_type*.
- Should be called with a final RETURN statement at the end with no arguments.

All functions must have a return type. If you do not have a return type, specify VOID.

## PL/pgSQL – Returning from a Function – Example

The following example implements RETURN NEXT and then a final RETURN after iterating through the rows:

```
CREATE OR REPLACE FUNCTION getAllStores()
RETURNS SETOF store AS
$BODY$
DECLARE r store%rowtype;
BEGIN
FOR r IN SELECT * FROM store WHERE storeid > 0 LOOP
    -- can do some processing here
    RETURN NEXT r;           ← RETURN NEXT statement
    -- return current row of SELECT
END LOOP;
RETURN;                      ← Final RETURN statement
END
$BODY$
LANGUAGE plpgsql;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

68

The current implementation of RETURN NEXT stores the entire result set before returning from the function. This means that if a PL/pgSQL function produces a very large result set, performance might be poor.

Data will be written to disk to avoid memory exhaustion, but the function itself will not return until the entire result set has been generated.

Currently, the point at which data begins being written to disk is controlled by the work\_mem configuration variable. Administrators who have sufficient memory to store larger result sets in memory should consider increasing this parameter.

## PL/pgSQL – Conditionals

The IF/THEN/ELSE block is defined as follows:

```
IF boolean-expression THEN
    statements
[ ELSIF boolean-expression THEN
    statements ]
[ ELSIF boolean-expression THEN
    statements ]
[ ELSE IF boolean-expression THEN
    statements ]
[ ELSE statements ]
END IF;
```



**Note:** Put the most commonly occurring condition first!

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

69

The IF/THEN/ELSE conditional is used in flow control to determine which statements are called based on evaluated expressions. Use the most commonly occurring condition first when implementing flow control with the IF/THEN/ELSE block.

## PL/pgSQL – Simple Loops

Loop conditionals:

- Continue to loop through statements until a condition is met
- Can be any of the following:
  - **LOOP**
  - **WHILE**
  - **EXIT**
  - **FOR**
  - **CONTINUE**

Loop statements are defined as follows:

```
LOOP SYNTAX:  
[ <<label>> ]  
LOOP  
    statements  
END LOOP [ label ];
```

Pivotal.

Loop conditionals will continue to loop through statements until some condition is met. A loop can be any of the following:

- **LOOP** – This defines an unconditional loop that is repeated indefinitely until an EXIT or RETURN statement is encountered.
- **EXIT** – This exits the inner most loop or block if no label is specified or an named outer loop if the label was defined for the loop or block.
- **CONTINUE** – The next iteration of an inner most loop is started if no label is provided. If a label is provided, it acts on the loop specified.
- **WHILE** – This repeats a sequence of statements until a condition has been met and the expression evaluates to true.
- **FOR** – Statements within the body of the FOR loop are executed over a range of integer values provided in the FOR statement. Once the last value in the range has been reached, the iteration stops.

## Simple Loops – LOOP and EXIT Example

```
LOOP
    -- some computations
    IF count > 0
        THEN EXIT;
        -- exit loop
    END IF;
END LOOP;
LOOP
    -- some computations
    EXIT WHEN count > 0; ←
    -- same result as previous example
END LOOP;
BEGIN
    -- some computations
    IF stocks > 100000
        THEN EXIT;
        -- causes exit from the BEGIN block
    END IF;
END;
```

The diagram illustrates the flow of control in the provided PL/SQL code. A large bracket on the right side groups the first two loops under the label "LOOP/END LOOP statements". Two arrows point from the "EXIT" statements in the second loop and the "BEGIN" block back to the "EXIT statement" label, indicating that both types of exits lead to the same point after the loop.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

71

Blocks or loops can be labeled to make it easier to reference them later in the code. The following conditions occur when working with loops:

- If no label is given, the innermost loop is terminated and the statement following `END LOOP` is executed next.
- If a label is given, it must be the label of the current or some outer level of nested SQL Procedural Language or block. Then the named loop or block is terminated and control continues with the statement after the corresponding `END` statement.
- If `WHEN` is specified, the loop exits occurs only if the boolean expression is true. Otherwise, control passes to the statement after `EXIT`.
- `EXIT` can be used with all types of loops and blocks. It is not limited to use with unconditional loops.

When used with a `BEGIN` block, `EXIT` passes control to the next statement after the end of the block.

## Simple Loops – CONTINUE Example

The following is an example of a CONTINUE loop. The innermost loop is called while count is < 50 and exits when count is > 100.

```
LOOP
    -- some computations
    EXIT WHEN count > 100;
    CONTINUE WHEN count < 50; ← CONTINUE statement
    -- some computations for count IN [50 .. 100]
END LOOP;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

72

When working with CONTINUE statements:

- If no label is given, the next iteration of the innermost loop is begun. All statements remaining in the loop body are skipped and control returns to the loop control expression, if any, to determine whether another loop iteration is needed.
- If the label is present, it specifies the label of the loop whose execution will be continued.

If WHEN is specified, the next iteration of the loop is begun only if the boolean expression is true. Otherwise, control passes to the statement after CONTINUE.

CONTINUE can be used with all types of loops; it is not limited to use with unconditional loops.

## Simple Loops – WHILE Loop Example

The following is an example of a WHILE loop. The loop iterates until customerid is equal to 50.

```
WHILE customerid < 50 LOOP  
    -- some computations  
END LOOP;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

73

The WHILE statement repeats a sequence of statements so long as the boolean expression evaluates to true. However, you must control the condition by incrementing or decrementing the condition. The expression is checked just before each entry to the loop body.

## Simple For Integer Loops

The following iterates through statements when *i* is  $\geq 1$  and  $\leq 3$ :

```
FOR i IN 1..3 LOOP
    -- i will take on the values 1,2,3 within the loop
END LOOP;
```

The following reverses the integer count:

```
FOR i IN REVERSE 3..1 LOOP
    -- i will take on the values 3,2,1 within the loop
END LOOP;
```

The following reverses the integer count and decrements *i* by 2:

```
FOR i IN REVERSE 8..1 BY 2 LOOP
    -- i will take on the values 8,6,4,2 within the loop
END LOOP;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

74

This form of FOR creates a loop that iterates over a range of integer values.

The variable name is automatically defined as type integer and exists only inside the loop. Any existing definition of the variable name is ignored within the loop. The two expressions giving the lower and upper bound of the range are evaluated once when entering the loop. If the BY clause is not specified, the iteration step is 1, otherwise, the value specified in the BY clause is used. The BY clause is evaluated once on loop entry.

If REVERSE is specified, then the step value is subtracted, rather than added, after each iteration.

If the lower bound is greater than the upper bound, or less than if REVERSE is used, the loop body does not execute. No error is raised.

If a label is attached to the FOR loop, then the integer loop variable can be referenced with a qualified name, using that label.

## Simple For Integer Loops

The following is an example of a `FOR` loop on queries. The loop iterates for each value retrieved from the `SELECT` statement.

```
CREATE FUNCTION nukedimensions() RETURNS integer AS $$  
DECLARE mdims RECORD;  
BEGIN  
    FOR mdims IN SELECT * FROM pg_class WHERE  
        schema='dimensions' LOOP  
        -- Now "mdims" has one record from pg_class  
        EXECUTE 'TRUNCATE TABLE ' || quote_ident(mdims.relname);  
    END LOOP;  
    RETURN 1;  
END;  
$$ LANGUAGE plpgsql;
```

FOR loop block

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

75

Using a `FOR` loop for queries, you can iterate through the results of a query and manipulate that data accordingly.

In a `FOR` loop that works on queries, if the loop is terminated by an `EXIT` statement, the last assigned row value is still accessible after the loop. The query used in this type of `FOR` statement can be any SQL command that returns rows to the caller. A `SELECT` statement is the most common case, but you can also use `INSERT`, `UPDATE`, or `DELETE` with a `RETURNING` clause. Some utility commands such as `EXPLAIN` will work too.

PL/pgSQL variables are substituted into the query text, and the query plan is cached for possible re-use.

## PL/pgSQL – Trapping Errors

Errors:

- Are generated by default if a function aborts execution
- Can be trapped and recovered from using a `BEGIN` block with an `EXCEPTION` clause, where:
  - The `SQLSTATE` variable is set to the error code
  - The `SQLERRM` variable is set to the error message

The syntax for trapping an error is as follows:

```
BEGIN
    statements
    EXCEPTION WHEN condition [OR condition ... ] THEN
        handler_statements
        [ WHEN condition [ OR condition ... ] THEN
        handler_statements ... ]
    END;
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

76

You can trap errors and recover from them by creating `BEGIN` blocks with an `EXCEPTION` clause.

A block containing an `EXCEPTION` clause is significantly more expensive to enter and exit than a block without one. Therefore, do not use `EXCEPTION` unless absolutely necessary.

Within an exception handler:

- The `SQLSTATE` variable contains the error code that corresponds to the exception that was raised.
- The `SQLERRM` variable contains the error message associated with the exception.

These variables are undefined outside of `EXCEPTION` handlers.

## PL/pgSQL – Common Exception Conditions

The following are a list of common error codes that can be generated:

Error Code	Description
CONNECTION_FAILURE	SQL cannot connect
DIVISION_BY_ZERO	Invalid division operation 0 in divisor
INTERVAL_FIELD_OVERFLOW	Insufficient precision for timestamp
NULL_VALUE_VIOLATION	Tried to insert NULL into NOT NULL column
RAISE_EXCEPTION	Error raising an exception
PLPGSQL_ERROR	PL/pgSQL error encountered at run time
NO_DATA_FOUND	Query returned zero rows
TOO_MANY_ROWS	Anticipated single row return, received more than 1 row

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

77

The slide shows a number of common error codes. A complete listing of error codes that can be captured are available in the PostgreSQL guide.

## User-defined Data Type

User-defined data types:

- Can be registered using the `CREATE TYPE` command
- Must use a unique name that is:
  - Distinct from any other data types in the same schema
  - Distinct from any table names in the same schema

The following is an example where a data type is defined:

```
CREATE TYPE queryreturn AS (
    customerid BIGINT,
    customerName CHARACTER VARYING,
    totalSalesAmt DECIMAL(12,2)
);
```

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

78

Often a function will need to return a data type other than one based on a table.

`CREATE TYPE` registers a new data type for use in the current database. The user who defines a type becomes its owner. If a schema name is given then the type is created in the specified schema. Otherwise it is created in the current schema. The type name must be distinct from the name of any existing type or domain in the same schema. The type name must also be distinct from the name of any existing table in the same schema.

## Lab: PostgreSQL Functions

In this lab you will create functions to perform simple, repeatable queries and tasks.

You will:

- Create SQL functions
- Create PL/pgSQL functions

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

79

In this lab you will create functions to perform simple, repeatable queries and tasks.

# Module 9: Developing Reports Using Advanced SQL

## Lesson 2: Summary

During this lesson the following topics were covered:

- Creating functions and procedures
- Using the functions and procedures to return values
- Implementing loops
- Setting traps for error handling

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

80

This lesson covered how to create and manage functions and procedures using a variety of programming languages available within the Greenplum environment. The various types of functions, input and output parameter definition, returning values, implementing loops, and setting traps for error handling were all discussed in the lesson.

# Module 9: Developing Reports Using Advanced SQL

## Lesson 3: Advanced SQL Topics and Performance Tips

In this lesson, you use commands to improve overall performance when manipulating data.

Upon completion of this lesson, you should be able to:

- Identify impacts updates and deletes have on performance
- Use specific data types to improve performance
- Avoid processing skew, data skew, and memory allocation problems using recommendations provided

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

81

Processing and data skew can have a strong effect on performance, especially when working with large data. An imbalance in how segments are loaded with data is directly impacted by how well you understand your data and how the data will be used. How you design for your data can directly impact performance, positively or negatively. Badly written SQL code or simply choosing one type of SQL command over another is another factor in poor performance. The end result is that you must learn to develop your code so that it does not, or reduces the chance, that it negatively impacts the performance for retrieving data.

In this lesson, you will:

- Identify impacts updates and deletes have on performance.
- Identify the link between data types and performance.
- Use specific data types to improve performance.
- Avoid processing skew, data skew, and memory allocation problems using recommendations provided.

## UPDATE and DELETE Performance

Consider the following when updating and removing rows from a table:

- Use `TRUNCATE TABLE` to delete all rows from a table
- Do not `DROP` the table and recreate the table
- To perform an `UPDATE` or `DELETE` on many rows:
  - Insert the rows into a temp table
  - Perform an `UPDATE JOIN` or a `DELETE JOIN`
  - Specify an equality in the `WHERE` clause between the distribution columns of the target table and all joined tables

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

82

There are performance considerations when performing inserts, updates and deletes in a data warehouse environments:

- When all rows from a table must be deleted, always use `TRUNCATE TABLE` rather than dropping and re-creating the table.
- To perform an update or delete to many rows in a table, insert the rows into a `TEMP` table then perform an `UPDATE JOIN` or `DELETE JOIN` with the `TEMP` table and base table. An `UPDATE JOIN` or `DELETE JOIN` will run an order of magnitude faster than single updates or deletes.

Keep in mind to perform an `UPDATE JOIN` or `UPDATE DELETE` the `WHERE` condition for the join must specify equality between corresponding distribution columns of the target table and all joined tables.

## Temporal and Boolean Data Types

To avoid:

- Processing skew, do not use temporal data types, such as date, time, timestamp, datetime, or interval, for distribution columns
- Data skew, never use a boolean data type, such as t or f for distribution columns

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

83

For some temporal formats, ordering of month and day in date input can be ambiguous. When possible, avoid using temporal data types for distribution columns to avoid processing skew.

If temporal data types are required, choose an optimal type. For example, use a DATE (4 bytes) if that is all that is needed rather than a TIMESTAMP (8 bytes). If you need both a DATE (4 bytes) and TIME (8 bytes), consider using a combined TIMESTAMP (8 bytes).

To avoid data skew, never use a boolean data type for distribution columns. Use a distribution key with unique values and high cardinality to distribute the data evenly across all segment instances.

## Avoid Approximate Numeric Data Types

Consider the following when defining numeric data types:

- Avoid floating point data types when possible
- Use the absolute minimum precision required
- Do not use approximate numerical types, such as double or float, for:
  - Distribution Columns
  - Join Columns
  - Columns that will be used for mathematical operations, such as SUM or AVG

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

84

Avoid the use of approximate numeric types, such as double or float, whenever possible to optimize performance in the Greenplum database.

Floating point data types inherently have performance implications in data warehouse environments. For example, a hash join can not be performed on a floating point data type rather a slower sort merge join is used.

Most importantly, do not use approximate numeric data types for distribution columns, join columns or columns that will be used for mathematical operations, such as SUM or AVG.

## Data Types and Performance

Consider the following when using data types:

- Use data types that will maximize performance and minimize:
  - Disk storage requirements
  - Scan time
- Choose the smallest integer type, such as `INTEGER` and not `NUMERIC(11, 0)`
- Choose integer types for distribution columns
- Choose the same integer types for join columns

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

85

It is important to use care when choosing data types.

Ideally, the data type used will:

- Minimize disk storage
- Minimize scan time thereby maximizing performance.

For numeric column data types, use the smallest data type to accommodate the data size. For example, use `SMALLINT` or `INT` instead of `BIGINT` to yield better performance and reduce storage space.

Choose integer types for distribution columns and choose the same integer types for commonly joined columns to maximize performance when joining tables.

## Use CASE to Avoid Multiple Passes

Consider the following when defining control logic:

- Use the CASE statement instead of DECODE, which is an encryption routine
- Put the most commonly occurring condition first
- Put the least commonly occurring condition last
- Always put a default (ELSE) condition should a condition not be met by previous statements

```
CASE
    WHEN state = 'CA'
        THEN state_population * 1.2
    ...
    WHEN state = 'WY'
        THEN state_population * 10.7
    ELSE
        0
END
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

86

The CASE statement is similar to the IF/THEN/ELSIF/ELSE block. DECODE is an encryption routine in Greenplum, whereas in Oracle, DECODE is used as part of a case statement.

When defining CASE or IF/THEN/ELSE blocks, put the most commonly occurring condition first and the least commonly occurring condition last. This reduces the amount of checks that have to be performed before a condition is met.

## SET Operation Cautions

Consider the following when using the SET operation:

- SET operations can improve performance
- Do not UNION a large numbers of queries against tables with millions or billions of rows as you can run low on memory and temp space
- Use a temporary table to store the results and populate it with separate queries
- Use a CASE statement, if possible, to retrieve the data in one pass.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

87

SET operations can improve performance overall. However, when using the SET operation, consider the following:

- Do not UNION a large numbers of queries against tables with millions or billions of rows. The optimizer will attempt to launch each query in parallel, resulting in running low or running out of memory and temp space.
- Use a temporary table to store the results and populate it with separate queries.
- Use a CASE statement, if possible, to retrieve the data in one pass.

## Avoid Multiple DISTINCT Operations – Part 1

The following is a common query which includes multiple distinct aggregate calls:

```
SELECT customerId,
       COUNT(DISTINCT transId),
       COUNT(DISTINCT storeId)
  FROM transaction
 WHERE transDate > '03/20/2008'
   AND transDate < '03/25/2008'
 GROUP BY 1
 ORDER BY 1
 LIMIT 100;
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

88

The slide shows a fairly common query for a report. BI tools like Microstrategy and Business Objects often tend to use this type of syntax.

While you may not be able to affect code generated in tools, you can develop cleaner and more effective ad-hoc queries. Running an explain of this plan shows GROUP AGGREGATE functions which can have performance impacts.

## Avoid Multiple DISTINCT Operations – Part 2

```
SELECT sub2.customerId,
       sub2.cd_transId,
       sub4.cd_storeId
  FROM (SELECT customerId, COUNT(transId) as cd_transId
            FROM ( SELECT customerId, transId
                  FROM transaction
                 WHERE transDate > '03/20/2008' AND transDate <
          '03/25/2008'
                  GROUP BY 1,2
             ) sub1
           GROUP BY 1
      ) sub2,
      (SELECT customerId, COUNT(storeId) as cd_storeId
            FROM ( SELECT customerId, storeId
                  FROM transaction
                 WHERE transDate > '03/20/2008' AND transDate <
          '03/25/2008'
                  GROUP BY 1,2
             ) sub3
           GROUP BY 1
      ) sub4
 WHERE sub4.customerId = sub2.customerId
 ORDER BY 1 LIMIT 100;
```

DISTINCTs replaced with subqueries so that large sorts are replaced with hash aggregates.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

89

Though it may not look as efficient, not having to sort on a huge table can have significant performance improvements.

Why is the second version faster? To execute the first query, the optimizer executes scans and SORTs of the data for each COUNT (DISTINCT) combination. This leads to a very resource intensive operations.

An explain of this example shows HASH AGGREGATES. Remember, it is better to replace large SORTs with hash aggregates.

## IsDate Function

The following function behaves like the `IsDate` function found in other DBMS systems:

```
CREATE OR REPLACE FUNCTION public.isdate(text)
RETURNS boolean AS $BODY$
begin
    perform $1::date;
    return true;
exception when others then
    return false;
end
$BODY$
LANGUAGE 'plpgsql' VOLATILE;
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

90

This function is not in the system, but is an excellent one to have on hand. It is a good utility function that allows you to verify whether or not a specified date is valid.

## Using Arrays

With Greenplum, it is possible to have an `ARRAY` data type with the following conditions:

- Arrays may not be the distribution key for a table
- You may not create indexes on `ARRAY` data type columns

The following is an example of an array that stores OIDs of dimension objects:

```
CREATE TEMPORARY TABLE dimensionsOID AS (
SELECT ARRAY(SELECT c.OID
             FROM pg_class c,
                  pg_namespace n
            WHERE n.oid = c.relnamespace
              AND n.nspname = 'dimensions'
          AS oidArray)
DISTRIBUTED RANDOMLY;
```

Pivotal.

You can create an array, but you cannot index columns that are array data types.

© 2015 Pivotal Software, Inc. All rights reserved.

91

## Update Statistics, Analyze EXPLAIN Plan, Clean Database Objects

During the development process:

- VACUUM tables if tables experience updates and deletes
- Ensure statistics are up to date to provide the best query plans for the query and existing data
- Analyze the EXPLAIN plan and re-write code to minimize performance impacts with:
  - Large sorts
  - Nested join loops
- If necessary, use indexes for your tables, but test them first
- Obtain the size of database objects

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

92

There are numerous things that you can do to improve the performance on your system. Remember to establish a baseline and establish reasonable goals against service level agreements.

When developing your code:

- Use VACUUM on tables to reduce bloat that may occur if the tables are constantly updated or deleted.
- Ensure statistics are up to date on the tables to get the best query plans that the optimizer can provide.
- Analyze the EXPLAIN plan to assess where the greatest time is being spent and reduce those times if possible through re-writes, changing data types, or balancing data on segments. In general for code development, eliminate large sorts and nested join loops. These can be resource hogs.
- If you are using indexes, use them appropriately. Ensure that they are being used by the optimizer and that they do have a positive impact on performance.
- In general, check the size of database objects, such as indexes and tables.

## Module 9: Developing Reports Using Advanced SQL

### Lesson 3: Summary

During this lesson the following topics were covered:

- Identifying the impact updates and deletes have on performance
- Using specific data types to improve performance
- Avoiding processing skew, data skew, and memory allocation problems using recommendations provided

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

93

This lesson reviewed some of the performance topics discussed earlier in the course, but emphasized its impact with regards to more advanced functions and behaviors. Specific cases on the impact of updates and deletes on performance, selecting appropriate data types for performance enhancements, and avoiding processing, data skew, and memory allocation problems were all discussed in the lesson.

## Module 9: Summary

Key points covered in this module:

- Combined data from multiple tables using JOINs
- Used EXPLAIN and EXPLAIN ANALYZE to help the optimizer determine how to handle a submitted query
- Improved query performance by keeping statistics up to date and tuning the database for sampling size and error conditions
- Determined when it is best to use an index and what type of index to use
- Defined OLAP and listed the OLAP grouping sets available in Greenplum
- Used functions to manipulate and return data to a caller
- Improved query performance by following a number of performance enhancement tips

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

94

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

- Combine data from multiple tables using JOINs.
- Use EXPLAIN and EXPLAIN ANALYZE to help the optimizer determine how to handle a submitted query.
- Improve query performance by keeping statistics up to date and tuning the database for sampling size and error conditions.
- Determine when it is best to use an index and what type of index to use.
- Define OLAP and list the OLAP grouping sets available in Greenplum.
- Use functions to manipulate and return data to a caller.
- Improve query performance by following a number of performance enhancement tips.

## Course Summary

Key points covered in this course:

- Gained a basic understanding of data warehousing concepts and be able to describe PostgreSQL to develop a foundation for understanding the Greenplum solution.
- Learned about the Greenplum architecture and hardware solutions to support the architecture so that they understand how to properly implement a solution based on their company's business needs.
- Built and implemented a Greenplum software solution to manage the Greenplum environment and database through a variety of Greenplum utility tools and PSQL.
- Learned PostgreSQL and Greenplum specific SQL features and functions to manage data and optimize SQL query performance in a Greenplum database.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

95

Listed are the key points covered in this course.

# Thank You

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

96

# Pivotal

A NEW PLATFORM FOR A NEW ERA

This slide is intentionally left blank.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

98

# Appendix A: UNIX and PostgreSQL Primer

## Topics:

- UNIX Primer
- VI Editor Cheat Sheet
- PSQL Meta-Commands

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

1

## Appendix A: UNIX and PostgreSQL Primer

This appendix provides an overview of commonly used Linux, vi editor, and psql meta-commands.

## UNIX Primer – Directory Locations

Path	Description
.	Current path
..	Parent directory
~	Current user's home directory
<code>~userid</code>	Home directory of the user specified
<code>\$MASTER_DATA_DIRECTORY</code>	Data directory for the master server – in this course, it is <code>/data/master/gpseg-1</code>
<code>\$GPHOME</code>	Base directory for Greenplum Database installation – in this course, it is <code>/usr/local/greenplum-db</code>

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

2

## UNIX Primer – Directory Commands

Command	Description
cd [directory]	Change to a specified directory. If no directory is specified, change to the home directory
pwd	Print working or current directory
ls [directory]	Obtain a listing of a directory
mkdir directory	Create the specified directory
rmdir directory	Delete the specified empty directory



### Example: Navigating and listing directories

```
[gpadmin@mdw ~]$ ls . # List the current directory  
[gpadmin@mdw ~]$ ls # List the current directory  
[gpadmin@mdw ~]$ ls /home/gpadmin # List the contents of /home/gpadmin  
[gpadmin@mdw ~]$ ls ~ # List the contents of the current user's home dir  
[gpadmin@mdw ~]$ ls ~root # List the contents of root's home directory  
[gpadmin@mdw ~]$ cd $MASTER_DATA_DIRECTORY # Change to the directory  
specified in the variable
```



**Note:** A comment on the UNIX command line is preceded by the # symbol.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

3

## UNIX Primer – File Access and Manipulation

Command	Description
cat [filename]	Display the content of one or more files on the screen
more [filename]	Display the content of one or more files a page at a time
mv filename new_filename	Rename or move the file
rm [-r] filename	Remove one or more files (recursively with the -r option)



### Example: Accessing and creating files

```
[gpadmin@mdw ~]$ cat hosts      # List the contents of the hosts file
mdw
smdw
[gpadmin@mdw ~]$ cat > hosts_new  # Redirect standard input to the file
sdw1
sdw2
^D                  # Use CTRL-D to end input
[gpadmin@mdw ~]$ more hosts_new
sdw1
sdw2
[gpadmin@mdw ~]$ mv hosts_new hosts_seg    # Rename hosts_new to hosts_seg
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

4

## UNIX Primer – Miscellaneous Commands

Command	Description
<code>ssh hostname</code>	Open a secure shell to the specified hostname
<code>scp filename hostname:filename</code>	Secure copy a file from the current system to the specified hostname and path
<code>gpssh -h hostname -e 'command'</code>	Greenplum secure shell to the specified hostname
<code>gpscp -h hostname filename =: path/filename</code>	Greenplum secure copy (there is no space between =: and the path/filename)
<code>tail filename</code>	Print the last 10 lines of a file to standard output
<code>tail -n filename</code>	Print the last n lines of a file to standard output



### Example: Copying files using gpscp

```
[gpadmin@mdw ~]$ gpscp -h smdw $MASTER_DATA_DIRECTORY/pg_hba.conf  
=:${MASTER_DATA_DIRECTORY}/pg_hba.conf # Copy pg_hba.conf to standby
```



### Example: Copying files using scp

```
[gpadmin@mdw ~]$ scp $MASTER_DATA_DIRECTORY/pg_hba.conf  
smdw:${MASTER_DATA_DIRECTORY}/pg_hba.conf # Copy pg_hba.conf to standby
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

5

# UNIX Primer – Miscellaneous Commands

## (Cont)



Example: Use gpssh to check file system size and the last 3 lines of today's logs

```
[gpadmin@mdw ~]$ cat hosts_seg
sdw1
sdw2
[gpadmin@mdw ~]$ gpssh -f hosts_seg -e 'df -k; gplogfilter -n 3
/data/primary/*/pg_log/gpdb-'`date +%Y-%m-%d`'* *.csv'
[sdw1] df -k;gplogfilter -n 3 /data/primary/*/pg_log/gpdb-2012-01-31*_.csv
[sdw1] Filesystem      1K-blocks   Used   Available  Use% Mounted on
[sdw1] /dev/mapper/VolGroup00-LogVol00
[sdw1]           16220072  3577324  11805528  24% /
[sdw1] /dev/sda1        101086    12660    83207  14% /boot
[sdw1] /dev/sdb1        52909824  35143352  15078800  70% /data
[sdw1] tmpfs            513440        0    513440   0% /dev/shm
[sdw1] requested timestamp range from beginning of data to end of data
[sdw1] ----- /data/primary/gpseg0/pg_log/gpdb-2012-01-31_000000.csv -----
-----
[sdw1] 2012-01-31 09:57:08.886117 EST|||p15824|th-1294956864||||0|||seg-
1|||||WARNING: |01000|Error when sending file rep stats to perfmon -
only -1 bytes of 392 sent|||||||0||cdbfilerep.c|4109|
...
[sdw2] df -k;gplogfilter -n 3 /data/primary/*/pg_log/gpdb-2012-01-31*_.csv
...
```

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

6

# Vi Cheat Sheet

Mode	Description		
Insert	In insert mode, you insert text into the text editor		
Command	Command mode allows you to enter vi commands, such as starting an insert, append, removing characters, words, and lines, or navigating through the contents of the file. Hit ESC to access Command mode.		
Command Mode Commands			
Command	Description	Command	Description
:x <i>file</i>	Save and quit	o	Open a new line after current line
:q	Quit without saving	O	Open a new line before current line
:w <i>file</i>	Save	r	Replace character at cursor
:wq <i>file</i>	Save and quit	R	Replace characters starting at cursor
:q!	Force quit without saving	:n	Go to line <i>n</i>
i	Insert before cursor	G	Go to last line of file
I	Insert at start of line	1G	Go to first line of file
a	Append after cursor	0	Move to beginning of the line
A	Append after end of line	\$	Move to the end of a line
x	Delete character at cursor	dd	Delete current line

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

7

## Vi Cheat Sheet (Cont)

### Command Mode Commands

Command	Description	Command	Description
yy	Yank (or cut) the current line	w	Move to the beginning of the next word
p	Paste after the cursor (or current line)	W	Move to the beginning of the next space-delimited word
P	Paste before the cursor (or current line)	b	Move to the beginning of the previous word
/string	Search forward for the string	B	Move to the beginning of the previous space-delimited word
?string	Search backward for the string	e	Move to end of the current word
n	Search for the next instance of the string	E	Move to the end of the current space-delimited word
N	Search for the previous instance of the string	k (up arrow)	Move up a line
u	Undo last change	j (down arrow)	Move down a line
U	Undo all changes to a line	l (right arrow)	Move to the right one character
.	Repeat last command	h (left arrow)	Move to the left one character

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

8

## PSQL Meta-Commands

Command	Description	Command	Description
\dt	List tables in current search path	\h	Help for SQL commands
\dn	List schemas	\q	Quit psql
\dtS	List system tables, including	\d+	Detailed information on an object
\dtS	List views, tables, and sequences	\dg \du	List roles
\dx	List external tables	\dv	List views
\l	List databases	\timing	Toggle timing of commands
\?	Help for meta-commands	\c <i>database</i>	Connect to the specified database

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

9

This slide is intentionally left blank.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

10

# Appendix B: Greenplum and Hadoop Integration

This appendix describes how to integrate Hadoop with a pre-existing Greenplum environment.

Upon completion of this module, you should be able to:

- Use Hadoop Distributed File System tables in a Greenplum database.
- Grant privileges on the HDFS protocol
- Create a Greenplum external table and populate it with HDFS data

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

1

## Appendix B: Greenplum and Hadoop Integration

Data co-processing of structured and unstructured data with Greenplum Database and Hadoop respectively is possible by integrating the two databases and taking advantage of the parallel processing architecture inherent in Greenplum Database.

Greenplum Database supports integration of a Hadoop Distributed File System by providing access to the gphdfs protocol to read and write unstructured data.

In this module, you will:

- Integrate Greenplum Database to the Hadoop Distributed File System.
- Create an external table from Greenplum Database to files on a pre-existing Hadoop file system.

## Summary of Steps for Hadoop Integration

To integrate a pre-configured Hadoop environment with a pre-existing Greenplum Database, the following must be performed:

- Set environmental variables required for integration process
- Set Greenplum configuration parameters for working with Hadoop
- Grant privileges to HDFS protocol

Once configured, you can create a Greenplum external table to content in Hadoop

Pivotal

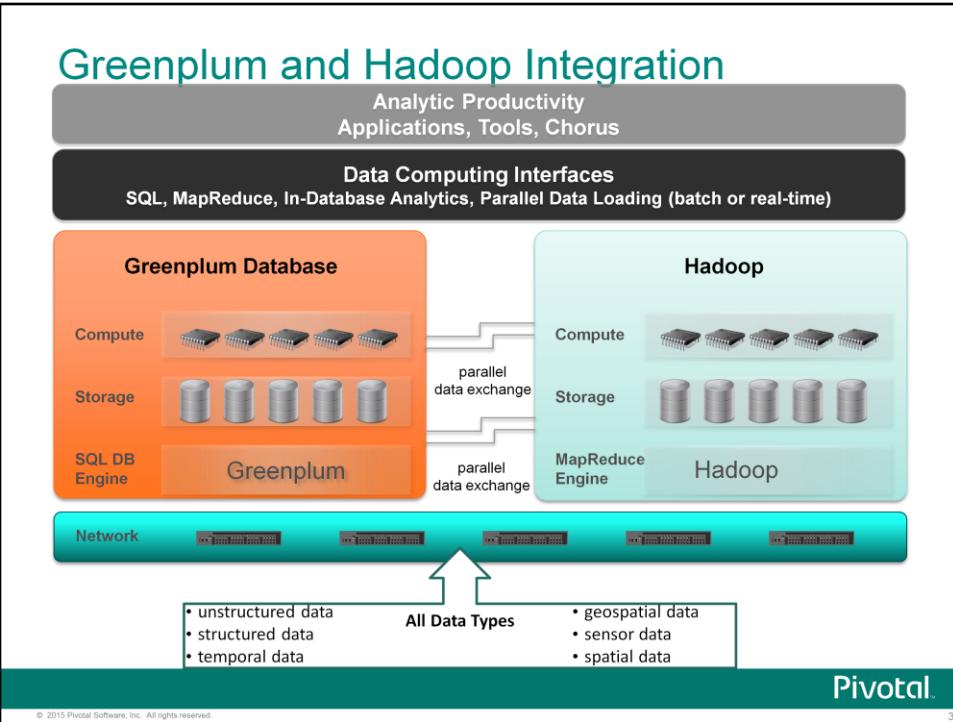
© 2015 Pivotal Software, Inc. All rights reserved.

2

To integrate Greenplum to Hadoop Distributed File System you must:

- Configure several environmental variables
- Modify several Greenplum configuration parameters required for Hadoop integration
- Perform a one-setup and grant privileges for the HDFS protocol.
- Identify and execute on-time setup
- Grant privileges for the HDFS protocol

Once the integration between the two databases has been configured, you can create a Greenplum external table to content that resides on the Hadoop Distribute File System.



## Greenplum and Hadoop Integration

Greenplum Database and Hadoop are complementary technologies that deliver a powerful solution for the analytics of structured, semi-structured, and unstructured data.

Some benefits of integrating the two technologies are listed below.

- Perform complex, high-speed, interactive analytics using Greenplum Database.
- Analytic Productivity, applications, Tools, and Chorus can be used on HDFS (Hadoop Distributed File System) data
- Data computing interfaces such as SQL and MapReduce
- Stream the data directly from Hadoop into Greenplum Database to incorporate unstructured or semi-structured data in the above analyses within Greenplum Database
- Hadoop can also be used to transform unstructured and semi-structured data into a structured format that can then be fed into Greenplum Database for high speed, interactive querying

## Set Environmental Variables

Example of `.bash_profile` for the user gpadmin

```
gpadmin@mdw:~$ .bash_profile
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin

export PATH

GPHOME=/usr/local/greenplum-db
export GPHOME
MASTER_DATA_DIRECTORY=/data/gphd_master/gpsne-1
export MASTER_DATA_DIRECTORY
PGDATABASE=gpadmin
export PGDATABASE
source $GPHOME/greenplum_path.sh

export JAVA_HOME=/usr/java/latest
export HADOOP_HOME=/usr/lib/gphd/hadoop
".bash_profile" 24L, 501C
```

Install the Hadoop client in the Greenplum cluster

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved. 4

## Set Environmental Variables

You will need to set several environmental variables for the `gpadmin` user in your Greenplum environment.

From the Greenplum master server, add the following lines to the `.bash_profile` of the `gpadmin` user:

- `export JAVA_HOME=/usr/java/latest`
- `export HADOOP_HOME=/usr/lib/gphd/hadoop`

The `JAVA_HOME` variable should be set to the location of java on the master server. The integration requires Java 1.6, which you will need to download and install on all Greenplum database hosts, including the master, standby, and segment hosts.

The `HADOOP_HOME` variable must point to the location of the Hadoop libraries, which are available from the Hadoop client. You will need to install the Hadoop client in the Greenplum cluster on all hosts before proceeding. This ensures access to all Hadoop libraries.

Server Configuration Parameters for Hadoop Targets			
Parameter	Available Value	Hadoop Distribution	Default Value
gp.hadoop_target_version	gphd-1.0	Greenplum HD 1.1/1.2 Pivotal HD 1.0/2.0 MapR 1.x, 2.x, 3.x MapR 4.x Cloudera 4.1-4.7 Cloudera 5.0/5.1	gphd-1.1
	gphd-1.1		
	gphd-1.2		
	gphd-2.0		
	gpmr-1.0		
	gpmr-1.2		
	cdh3u2		
	cdh4.1		
	hdp2		
gp.hadoop_home	The value stored in the \$HADOOP_HOME variable		NULL

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

5

## Server Configuration Parameters for Hadoop Targets

After setting the parameters, you will need to update the following master server configuration parameters in the `postgresql.conf` file with the `gpconfig` command:

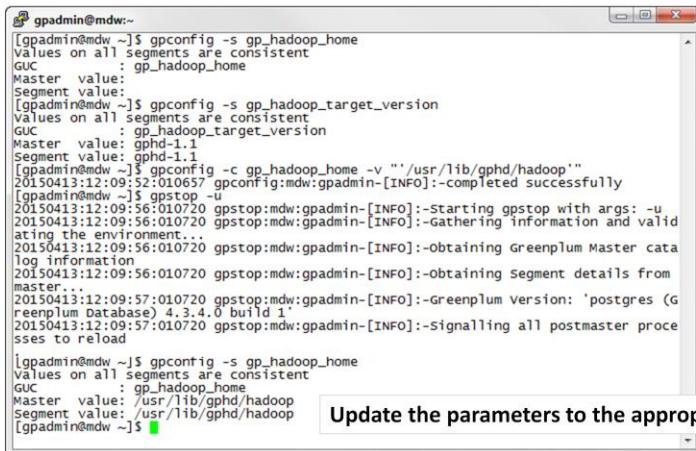
- `gp.hadoop_target_version`
- `gp.hadoop_home`

If not specified in the `postgresql.conf`, Greenplum will use the `gphd-1.1` library for the integration. If your distribution differs, select the appropriate value from the Available Value column.

The `gp.hadoop_home` parameter must be the same as the `HADOOP_HOME` variable set in the `.bash_profile` file. In this case, it is set to `/usr/lib/gphd/hadoop`.

After modifying the `postgresql.conf` file, you will need to re-read the `postgresql.conf` file with the `gpstop -u` command.

## Setting the Parameter Values



```
[gpadmin@mdw ~]$ gpconfig -s gp_hadoop_home
values on all segments are consistent
GUC      : gp_hadoop_home
Master value:
Segment value:
[gpadmin@mdw ~]$ gpconfig -s gp_hadoop_target_version
Values on all segments are consistent
GUC      : gp_hadoop_target_version
Master value: gphd-1.1
Segment value: gphd-1.1
[gpadmin@mdw ~]$ gpconfig -c gp_hadoop_home -v "/usr/lib/gphd/hadoop"
20150413:12:09:52:010657 gpconfig:mdw:gpadmin-[INFO]:-completed successfully
[gpadmin@mdw ~]$ gpstop -u
20150413:12:09:56:010720 gpstop:mdw:gpadmin-[INFO]:-Starting gpstop with args: -u
20150413:12:09:56:010720 gpstop:mdw:gpadmin-[INFO]:-Gathering information and validating the environment...
20150413:12:09:56:010720 gpstop:mdw:gpadmin-[INFO]:-obtaining Greenplum Master catalog information
20150413:12:09:56:010720 gpstop:mdw:gpadmin-[INFO]:-obtaining Segment details from master...
20150413:12:09:57:010720 gpstop:mdw:gpadmin-[INFO]:-Greenplum Version: 'postgres (Greenplum Database) 4.3.4.0 build 1'
20150413:12:09:57:010720 gpstop:mdw:gpadmin-[INFO]:-signalling all postmaster processes to reload
[gpadmin@mdw ~]$ gpconfig -s gp_hadoop_home
values on all segments are consistent
GUC      : gp_hadoop_home
Master value: /usr/lib/gphd/hadoop
Segment value: /usr/lib/gphd/hadoop
[gpadmin@mdw ~]$
```

Update the parameters to the appropriate values

Pivotal.

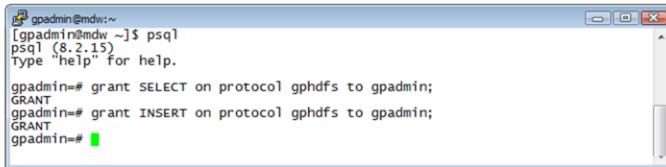
© 2015 Pivotal Software, Inc. All rights reserved.

6

## Grant Privileges for gphdfs Protocol

Usage: Granting read and write privileges to gpadmin

```
gpadmin=# GRANT SELECT ON PROTOCOL HDFS TO gpadmin;
gpadmin=# GRANT INSERT ON PROTOCOL HDFS TO gpadmin;
```



```
[gpadmin@mdw:~ [gpadmin@mdw ~]$ psql
psql (8.2.15)
Type "help" for help.

gpadmin=# grant SELECT on protocol gphdfs to gpadmin;
GRANT
gpadmin=# grant INSERT on protocol gphdfs to gpadmin;
GRANT
gpadmin=#
```



**Note:** The `SELECT` privilege allows `gpadmin` to create readable external tables with the `gphdfs` protocol. The `INSERT` privilege lets you create and use writable external tables with the `gphdfs` protocol.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

7

## Grant Privileges for gphdfs Protocol

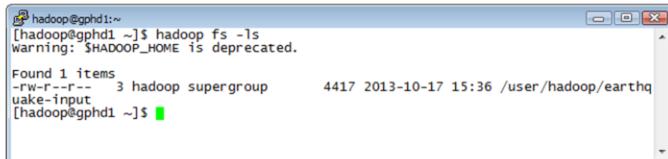
You must now provide read and write privileges on the gphdfs protocol to the `gpadmin` user. This lets the `gpadmin` user create readable and writeable external tables to content on HDFS using the gphdfs protocol.

From a pSQL session on to the Greenplum environment you are configuring for integration, execute the grant command for `SELECT` and `INSERT` privileges for the `gpadmin` user.

## Verify Access to Hadoop Content

Usage: Verify your Hadoop user can access and read file contents

```
[hadoop@gphd1 ~]$ hadoop fs -ls
```



```
[hadoop@gphd1:~]$ hadoop fs -ls
[hadoop@gphd1:~]$ hadoop fs -ls
Warning: $HADOOP_HOME is deprecated.

Found 1 items
-rw-r--r-- 3 hadoop supergroup      4417 2013-10-17 15:36 /user/hadoop/earthq
uake-input
[hadoop@gphd1:~]$
```

 **Note:** Verify you have access to the file being used to populate the Greenplum external table.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

8

## Verify Access to Hadoop Content

Before attempting to create an external table and read or write to the HDFS system, you should verify that your Hadoop user can access and read content from HDFS.

In this example, the `hadoop fs -ls` command is used to list the user-related content for the Hadoop user, `hadoop`.

## Read the HDFS Data

Usage: Granting Privileges to user gpadmin

```
[hadoop@gphd1 ~]$ hadoop fs -cat /user/hadoop/earthquake-input
```

```
[hadoop@gphd1:~]$ hadoop fs -cat earthquake-input
warning: SHADOOP_HOME is deprecated.

nc|wednesday, October 10, 2012 03:45:36 UTC|39.5662|-123.3917|1.8|8.90| 9|Northe
rn California
hv|wednesday, October 10, 2012 03:32:29 UTC|19.4028|-155.2697|2.9|1.90|24|island
of Hawaii, Hawaii
hv|wednesday, October 10, 2012 03:24:59 UTC|19.4048|-155.2673|2.6|2.10|17|island
of Hawaii, Hawaii
nn|wednesday, October 10, 2012 03:21:16 UTC|36.7553|-115.5388|1.2|7.00|20|Nevada
nn|wednesday, October 10, 2012 03:09:13 UTC|38.5830|-119.4507|1.3|7.00|7|Centra
l California
uw|wednesday, October 10, 2012 03:07:14 UTC|47.7083|-122.3250|2.0|29.70|36|Seatt
le-Tacoma urban area, Washington
ci|wednesday, October 10, 2012 02:52:38 UTC|32.8157|-116.1407|1.3|7.40|22|Southe
rn California
ci|wednesday, October 10, 2012 02:46:21 UTC|33.9320|-116.8478|1.8|7.60|87|Southe
rn California
hv|wednesday, October 10, 2012 02:17:29 UTC|19.4042|-155.2688|1.9|1.70|17|island
of Hawaii, Hawaii
```

 **Note:** Verify you can read the data that will be used to populate the Greenplum external table.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

9

## Read the HDFS Data

Read the HDFS data using the `hadoop fs -cat` command. By viewing the content of your data, you can verify how many and what type of columns you will use when creating the external table.

## Creating an External Table

Usage: Example of an external table creation

```
gpadmin=# CREATE EXTERNAL TABLE TABLE_NAME (FIELD NAMES)
LOCATION ('gphdfs://hdfs_host:port/path/file_name')
FORMAT 'TEXT' (DELIMITER '|');

gpadmin=#
gpadmin=#
gpadmin# create external table ext_hdfs (
gpadmin#   f1 text,
gpadmin#   f2 text,
gpadmin#   f3 float,
gpadmin#   f4 float,
gpadmin#   f5 float,
gpadmin#   f6 float,
gpadmin#   f7 integer,
gpadmin#   f8 text)
gpadmin# location
gpadmin# ('gphdfs://172.16.1.21:9000/user/hadoop/earthquake-input') format 'TEXT' (delimiter '|');
CREATE EXTERNAL TABLE
gpadmin=#
gpadmin=#

```

 **Hadoop IP Host** **Port number**

 **Note:** Know the field types you are going to create on your external table. A field omitted or wrongly defined can compromise your data retrieval.

Pivotal

© 2015 Pivotal Software, Inc. All rights reserved.

## Creating an External Table

The **gphdfs** protocol allows for a link between Greenplum and HDFS. Thus, a Greenplum external table can be populated using getting HDFS unstructured data. The data is read in parallel from HDFS into the Greenplum segments for a fast process.

Connect to your Greenplum Database environment to create an external table as shown.

## Accessing HDFS Data Loaded into Greenplum

```
gpadmin@mdw:~$ psql
psql (8.2.15)
Type "help" for help.

gpadmin# select count(*) from ext_hdfs ;
count
-----
47
(1 row)

gpadmin# psql
psql (8.2.15)
Type "help" for help.

gpadmin# select * from ext_hdfs limit 10;
f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8
----+---+---+---+---+---+---+---+
nc | Wednesday, October 10, 2012 03:45:36 UTC | 39.5662 | -123.3917 | 1.8 | 8.9 | 9 | Northern California
hv | Wednesday, October 10, 2012 03:32:29 UTC | 19.4028 | -155.2697 | 2.9 | 1.9 | 24 | Island of Hawaii, Hawaii
hv | Wednesday, October 10, 2012 03:24:59 UTC | 19.4048 | -155.2673 | 2.6 | 2.1 | 17 | Island of Hawaii, Hawaii
nn | Wednesday, October 10, 2012 03:21:16 UTC | 36.7553 | -115.5388 | 1.2 | 7 | 20 | Nevada
nn | Wednesday, October 10, 2012 03:09:13 UTC | 38.583 | -119.4507 | 1.3 | 7 | 7 | Central California
uw | Wednesday, October 10, 2012 03:07:14 UTC | 47.7083 | -122.325 | 2 | 29.7 | 36 | Seattle-Tacoma urban area, Washington
ci | Wednesday, October 10, 2012 02:52:38 UTC | 32.8157 | -116.1407 | 1.3 | 7.4 | 22 | Southern California
ci | Wednesday, October 10, 2012 02:46:21 UTC | 33.932 | -116.8478 | 1.8 | 7.6 | 87 | Southern California
hv | Wednesday, October 10, 2012 02:17:29 UTC | 19.4042 | -155.2688 | 1.9 | 1.7 | 17 | Island of Hawaii, Hawaii
ak | Wednesday, October 10, 2012 02:06:25 UTC | 60.5271 | -152.2992 | 1.3 | 83.9 | 7 | Southern Alaska
(10 rows)
gpadmin#
```

Once the HDFS data has been loaded into Greenplum, you can use it as regular Greenplum data

Pivotal.

© 2015 Pivotal Software, Inc. All rights reserved.

11

## Accessing HDFS Data Loaded into Greenplum

Once the data has been loaded into the Greenplum external table you can use it as regular Greenplum data.

# Pivotal

A NEW PLATFORM FOR A NEW ERA