Greenplum数据库v3.3
系统管理员培训

课程介绍

课程内容

1. 产品概述
2. 系统配置和验证
3. 软件安装和系统初始化
4. 客户端应用程序
5. 服务器配置
6. 数据定义语言(DDL)
7. 角色、权限和基于角色的资源队列
8. 表分区功能
9. 数据加载
10. 数据操作语言(DML)和数据查询语言(DQL)
11. 性能优化
12. 系统管理
13. 冗余和高可用性
14. 备份和恢复
15. 数据库内部结构

Greenplum

课程资料和安排

- 授课形式
- 培训指导手册
- 培训练习
  - 根据手册的练习内容
  - 需要网络和ssh接入
- Q & A讨论
- 反馈和课程评价

培训对象

数据库管理员(DBA)

- 基本的SQL语法
- 常规的数据库概念

系统管理员

- 基本的UNIX命令(cd, ls, ssh, cat, grep等）
- VI（文本编辑器）的使用

学习目标

- 理解Greenplum数据库的架构和组成部分
- 熟悉Greenplum数据库的特性和功能
- 懂得如何使用Greenplum数据库完成基本操作
- 掌握如何管理Greenplum数据库系统，以及解决故障

第1课

Greenplum产品概述

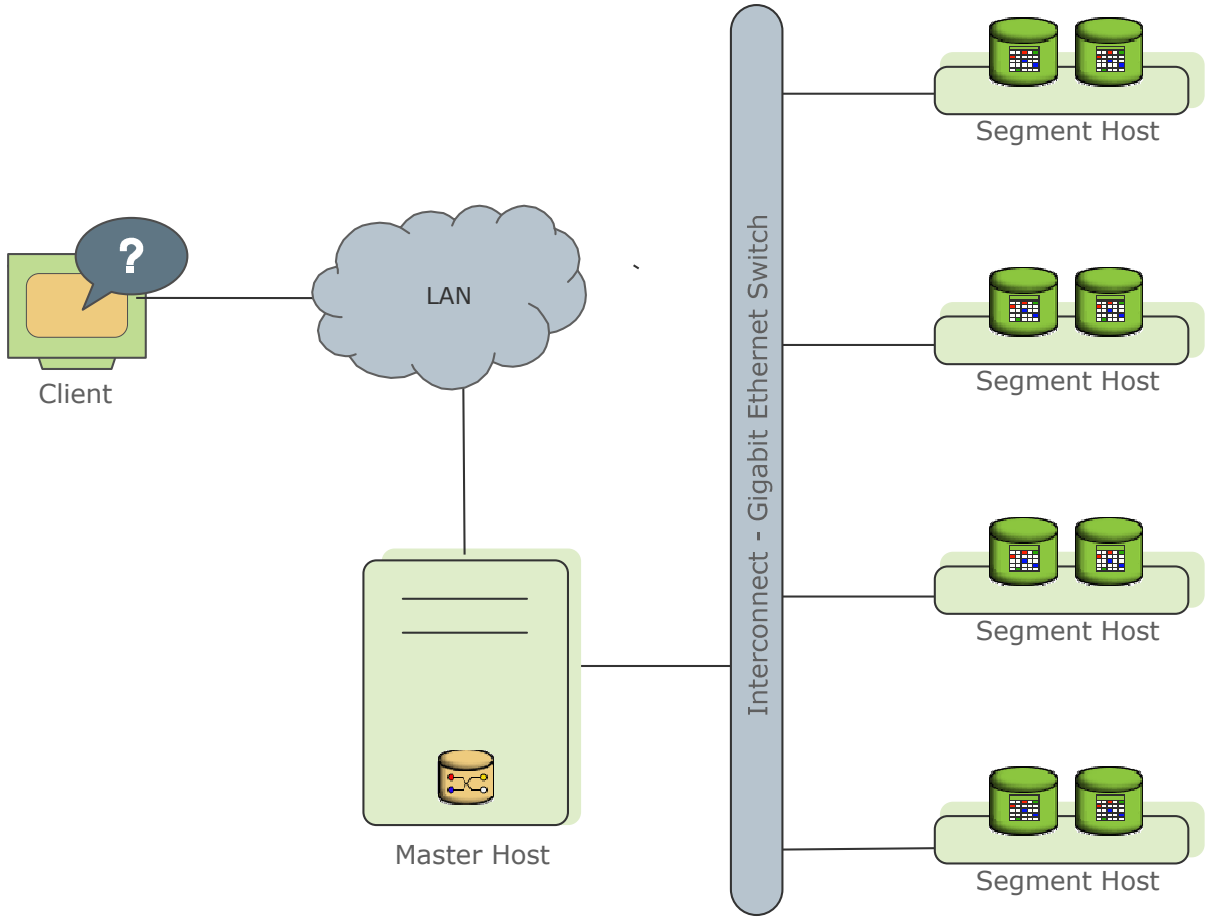
- 主要架构和组成部分
- 高可用性设计
- 系统管理内容
- 重要概念—数据分布和查询

Greenplum数据库

- **海量并行处理(MPP) DBMS**
- **基于 PostgreSQL 8.2**
  - 相同的客户端功能
  - 增加支持并行处理的技术
  - 增加支持数据仓库和BI的特性
    - 外部表/并行加载
    - 资源管理
    - 查询优化器增强

Greenplum

Greenplum基本架构



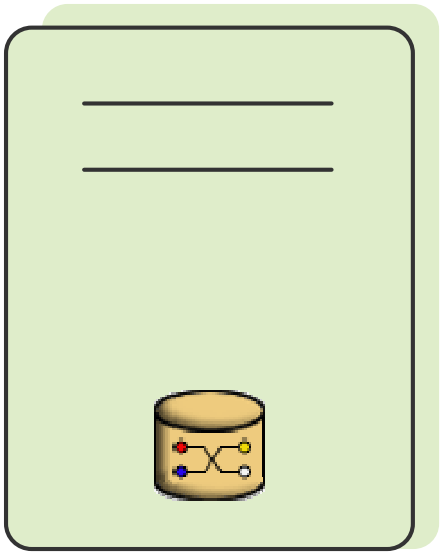
Greenplum

客户端程序

- psql
- pgAdmin III
- ODBC
- JDBC
- Perl DBI
- Python
- libpq

Greenplum

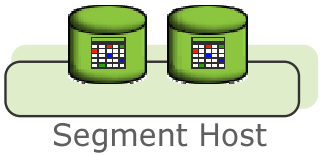
Master主机

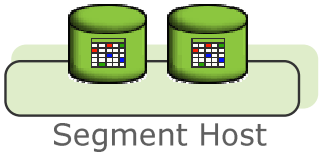


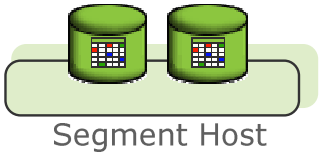
- 访问系统的入口
- 数据库侦听进程 (postgres)
- 处理所有用户连接
- 建立查询计划
- 协调工作处理过程
- 管理工具
- 系统目录表和元数据（数据字典）
- 不存放任何用户数据

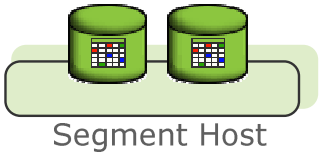
Greenplum

Segments主机





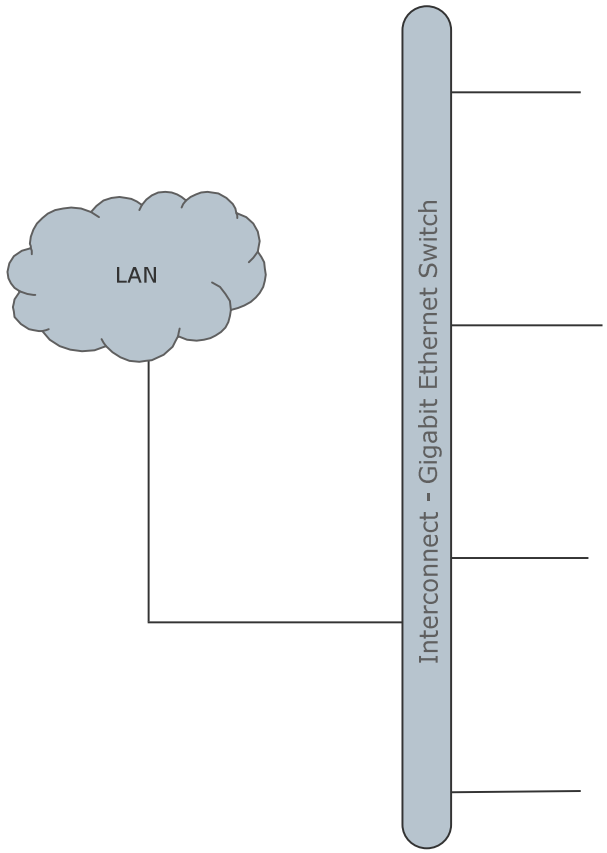




- 每个Segment上存放一部分用户数据
- 一台Segment主机可以有多个实例
- 通过增加Segment主机实现线性扩展
- 用户不能直接存取访问
- 所有对段的访问都经过Master
- 数据库监听进程(postgres)来自Master的连接

Greenplum

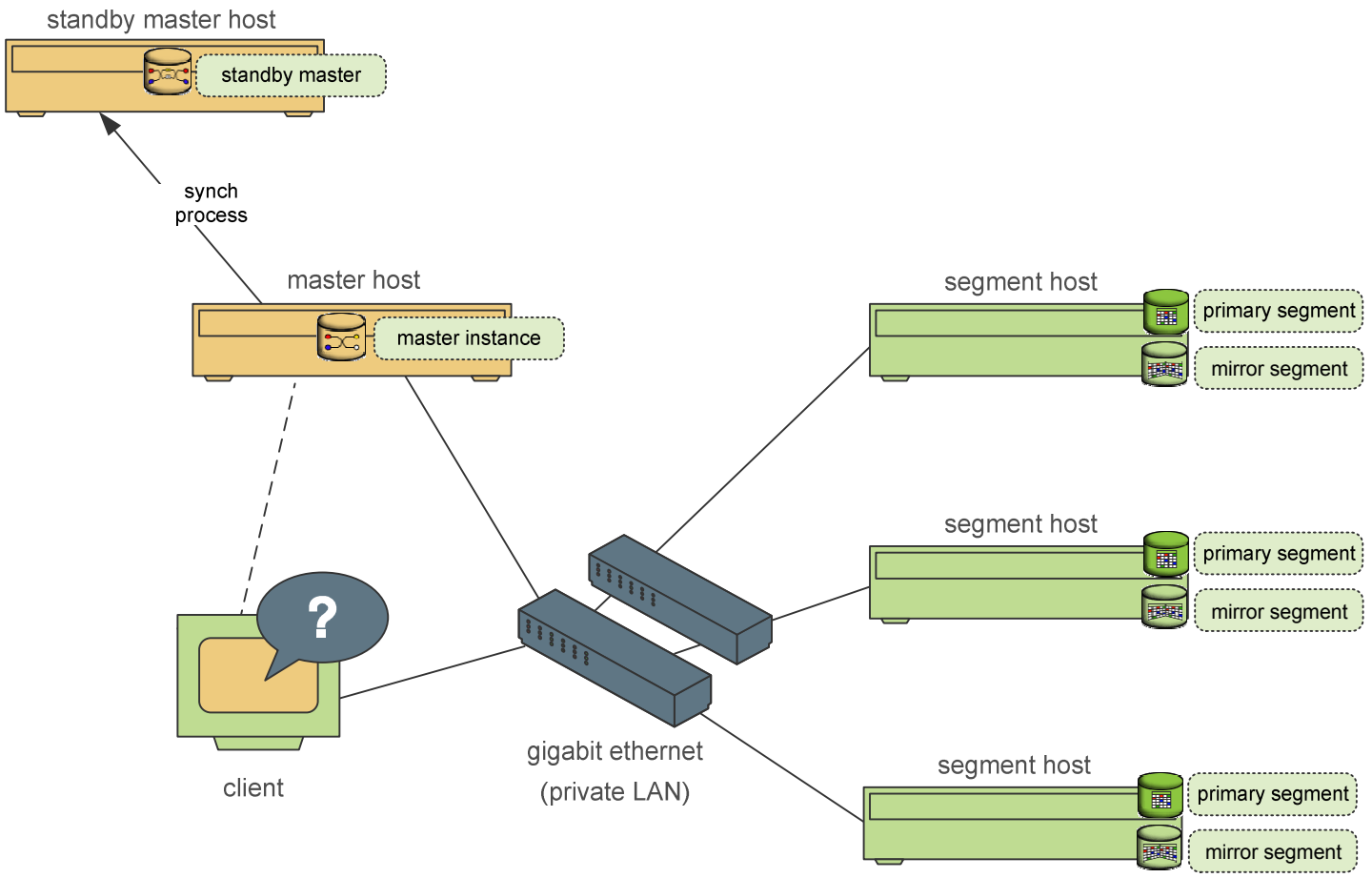
Interconnect



- Greenplum数据库之间的连接层
- 进程间协调和管理
- 基于千兆以太网架构
- 属于系统内部私网配置
- 支持两种协议：TCP or UDP

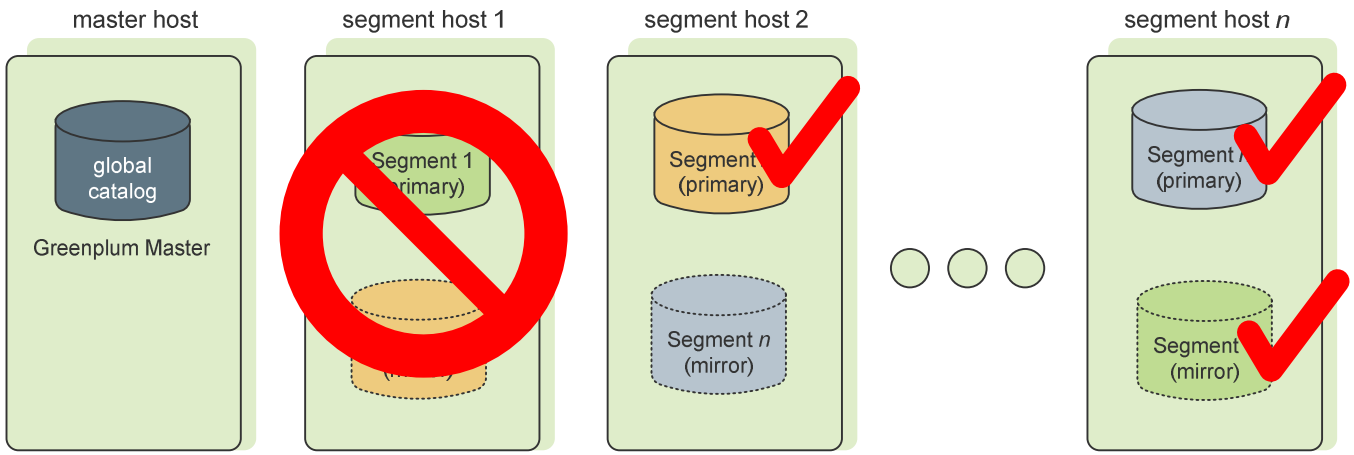
Greenplum

高可用性的架构



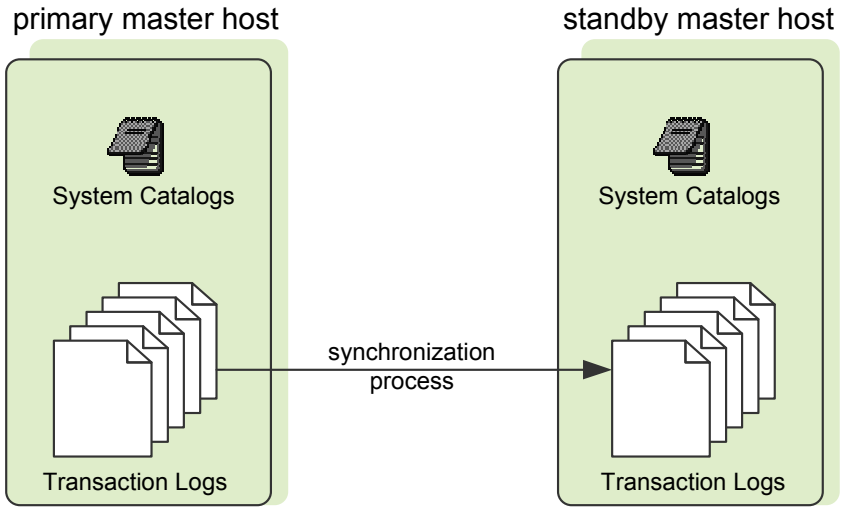
Greenplum

数据冗余 –Segment镜像



Greenplum

Master镜像—热备Master



Greenplum

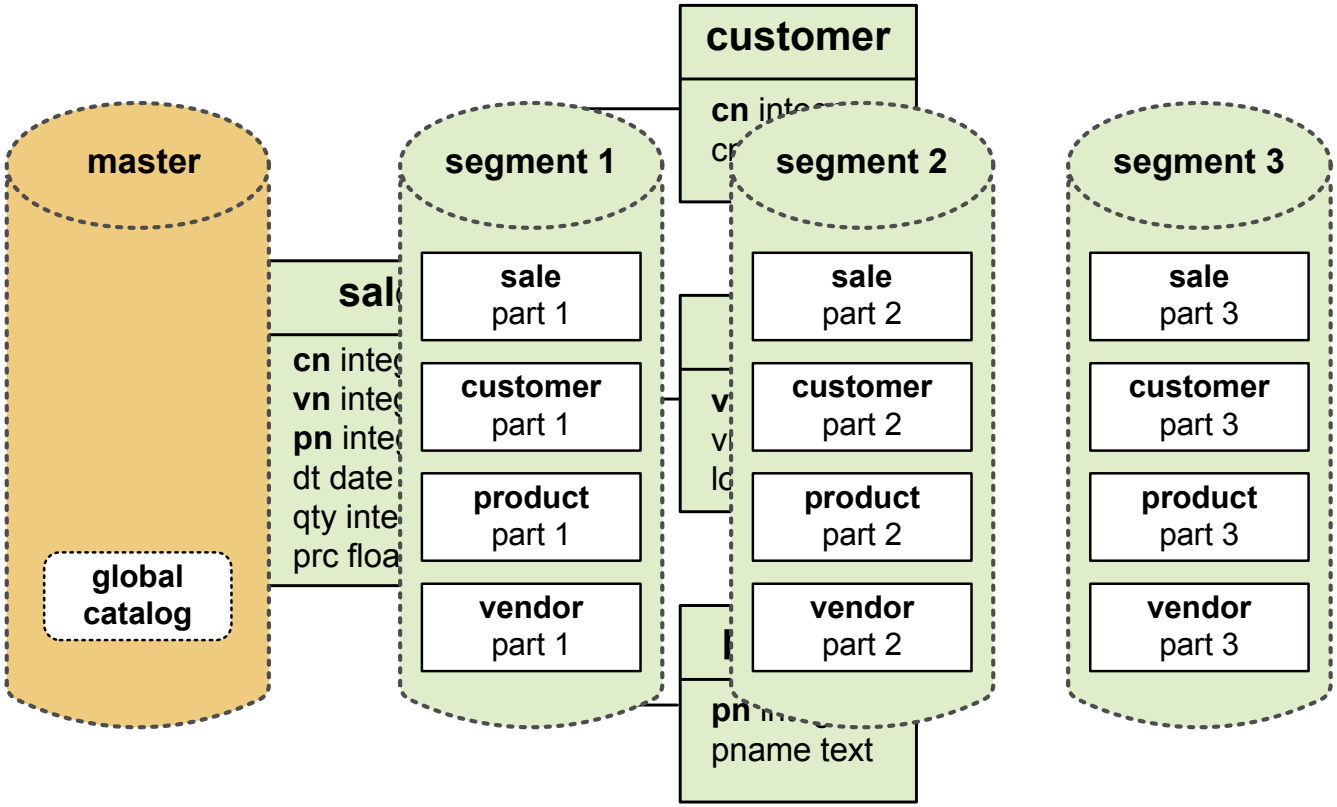
Greenplum 管理命令

- 存放位置$GPHOME/bin
- 命令行功能可提供：
  - 系统的设置和初始化
  - 系统的开启和关闭
  - 系统的扩容和数据的重新分布
  - 复原故障的Master或Segment实例
  - 备份和恢复
  - 为并行加载创建文件服务器
  - 系统状态报告
- 性能监控工具Performance Monitor

Greenplum数据库的重要概念

- 表分布方式
- 并行查询的实现方式

分布式的数据表

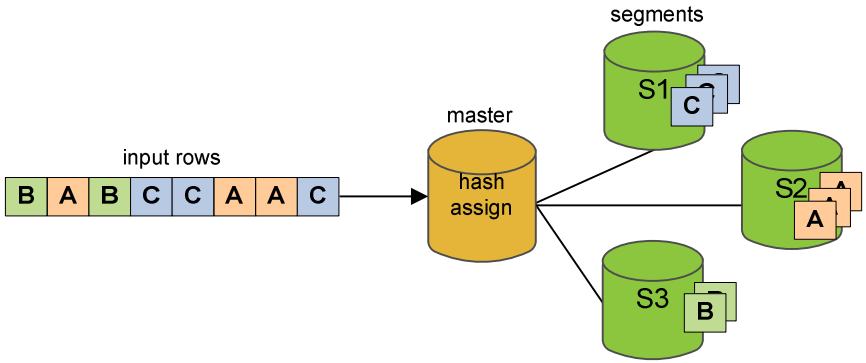


Greenplum

表分布的方式

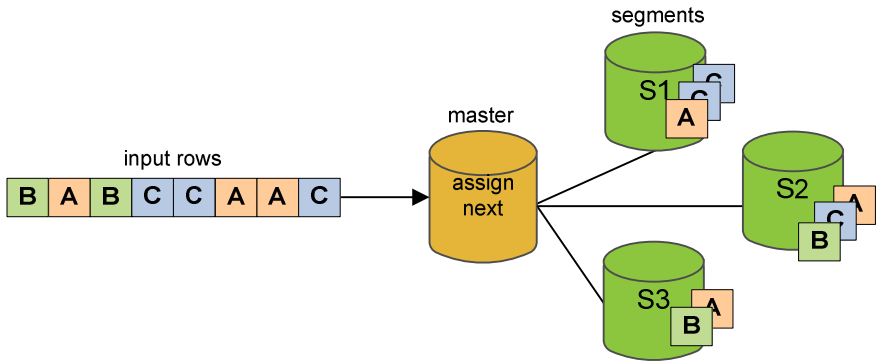
Hash分布

- `CREATE TABLE … DISTRIBUTED BY (column [,…])`
- 同样内容的键值被分配到同一个Segment上



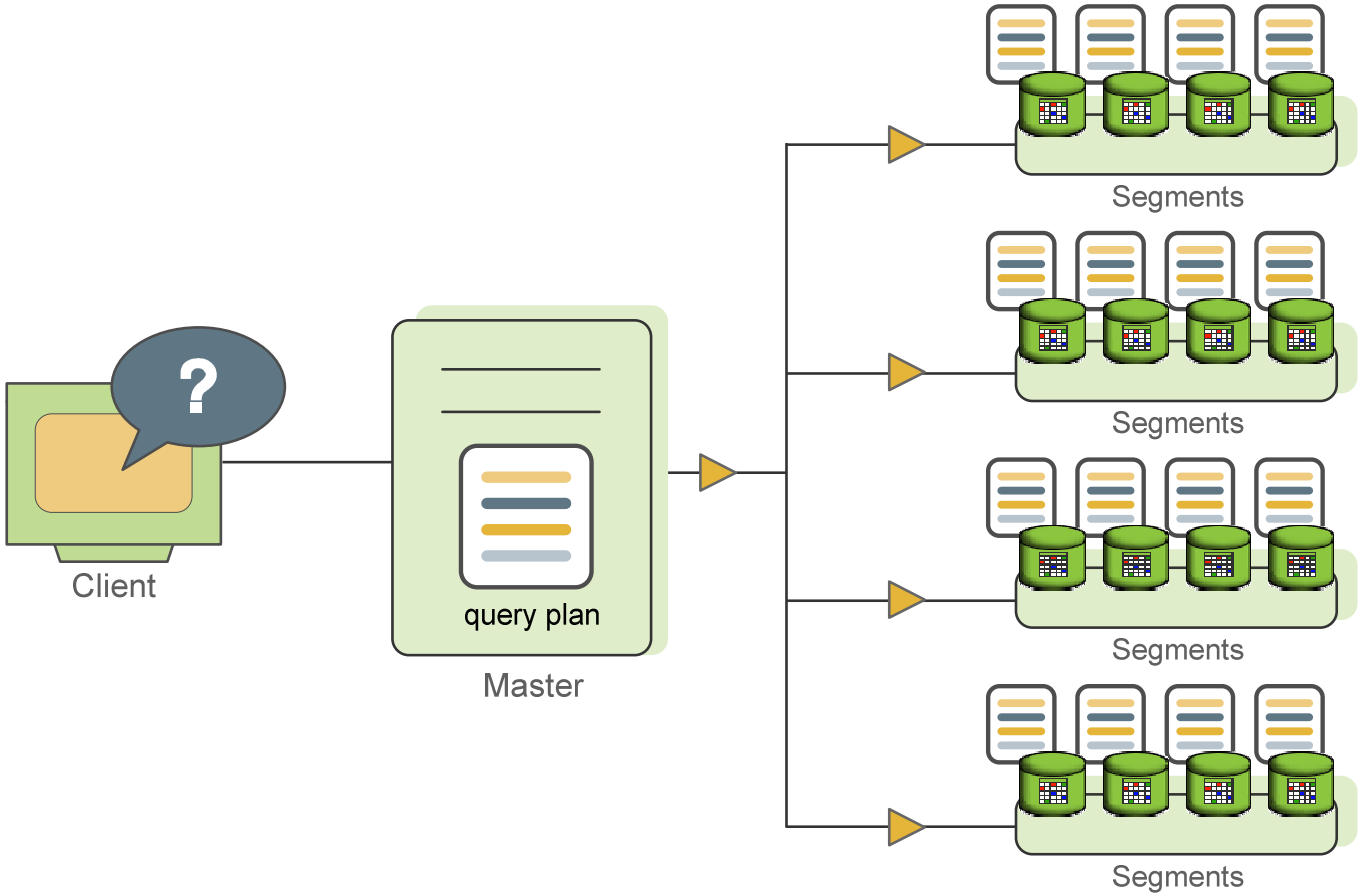
循环分布

- `CREATE TABLE … DISTRIBUTED RANDOMLY`
- 具有同样数值的行内容并不一定在同一个Segment上



Greenplum

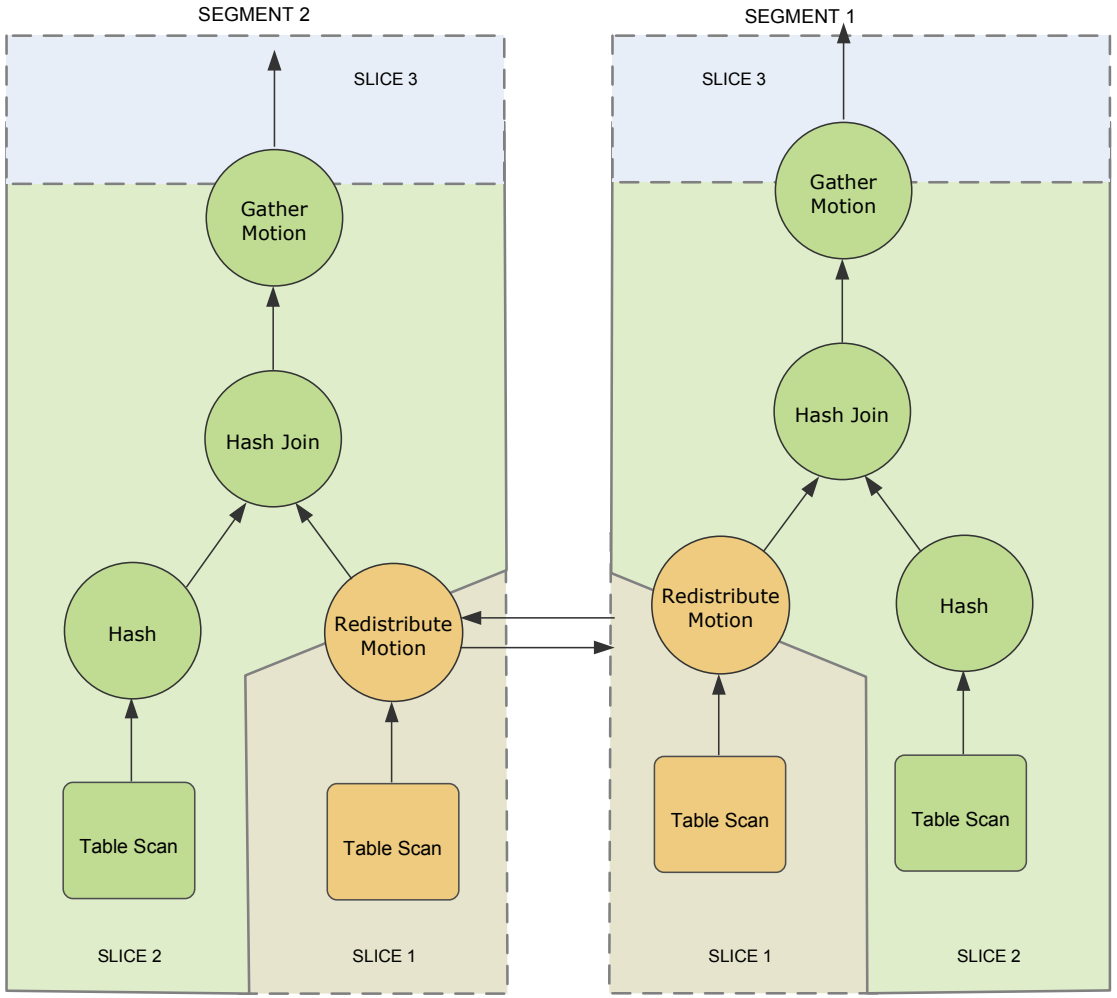
计划和调度一个查询



Greenplum

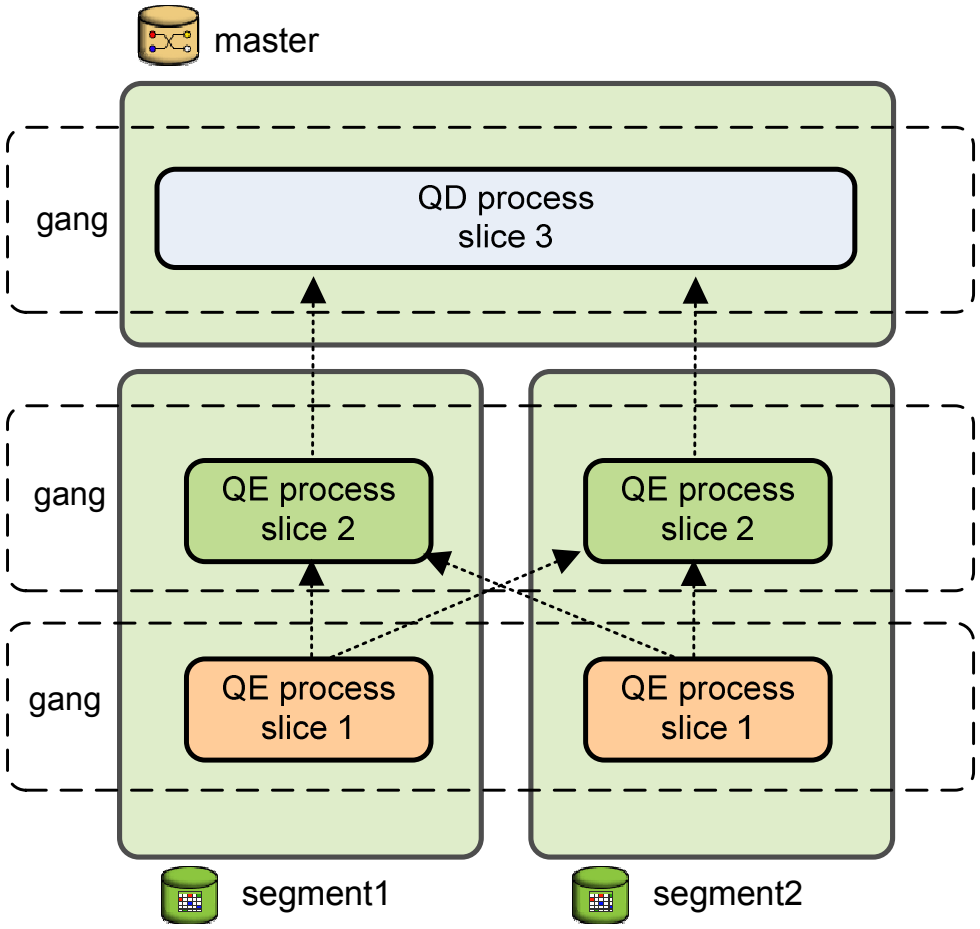
并行查询计划

```
SELECT customer, amount FROM sales JOIN customer USING (cust_id) WHERE date=04302008;
```



Greenplum

查询工作的进程



Greenplum

3.3版本的新增内容

- 在线的系统扩容方式
- 数据表再分布更新
- pgAdmin III工具

Greenplum

第1课练习

复习Greenplum概念

第2课

系统配置和验证

- Greenplum和硬件设备
- 参考架构
- OS内核优化
- Greenplum系统验证工具
- 估算存储容量

Greenplum软硬件需求

- **Greenplum—数据库软件方案**
  - 给客户多种硬件平台的选择
  - 数据库的性能与硬件性能有密切关系
  - Greenplum有多种型号的硬件平台经过测试认证
- **选择硬件的考虑因素**
  - Segment服务器具有相同的硬件配置
  - Master服务器需要较高的CPU和内存资源
- **认证的操作系统**
  - Solaris 10 update 4 /5 (添加推荐的patches)
  - SUSE Linux Enterprise 10
  - Red Hat Enterprise Linux 5.x
  - CentOS 5.x

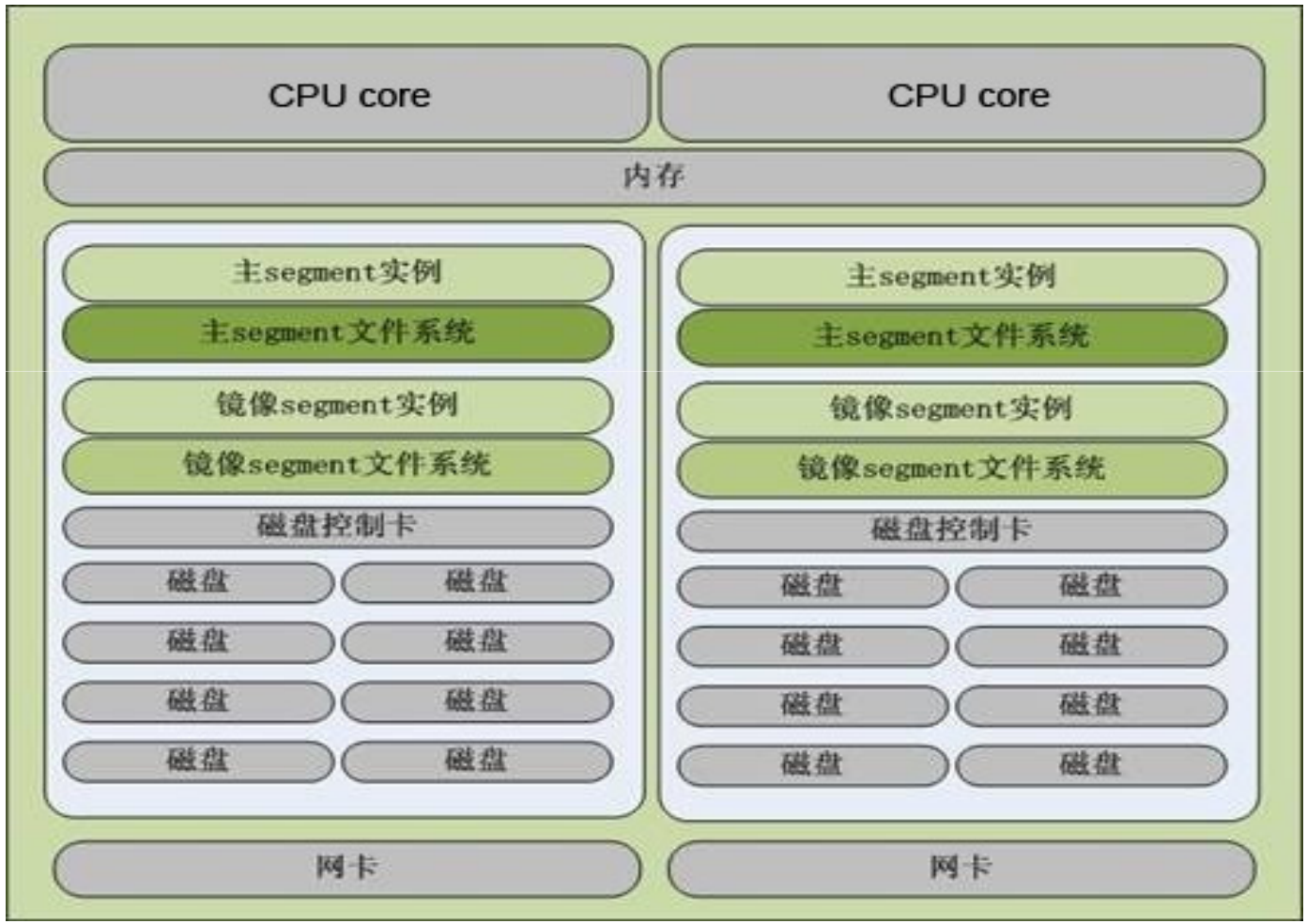
Greenplum

Greenplum硬件配置

- **Segment主机的推荐配置**
  - 2颗P3/Athlon CPU, dual-core
  - 32GB内存
  - 内置或直连的高速磁盘阵列
  - 4个以上的千兆网口
- **Greenplum认证的硬件型号（x86 PC服务器）**
  - Sun Fire x4240 (Master)+ Sun Fire x4540 (Segment)
  - HP DL180G6
  - Dell R710 + MD1120磁盘阵列
  - Dell 2950 + MD1000磁盘阵列
  - 华赛Oceanstor T3500

Greenplum

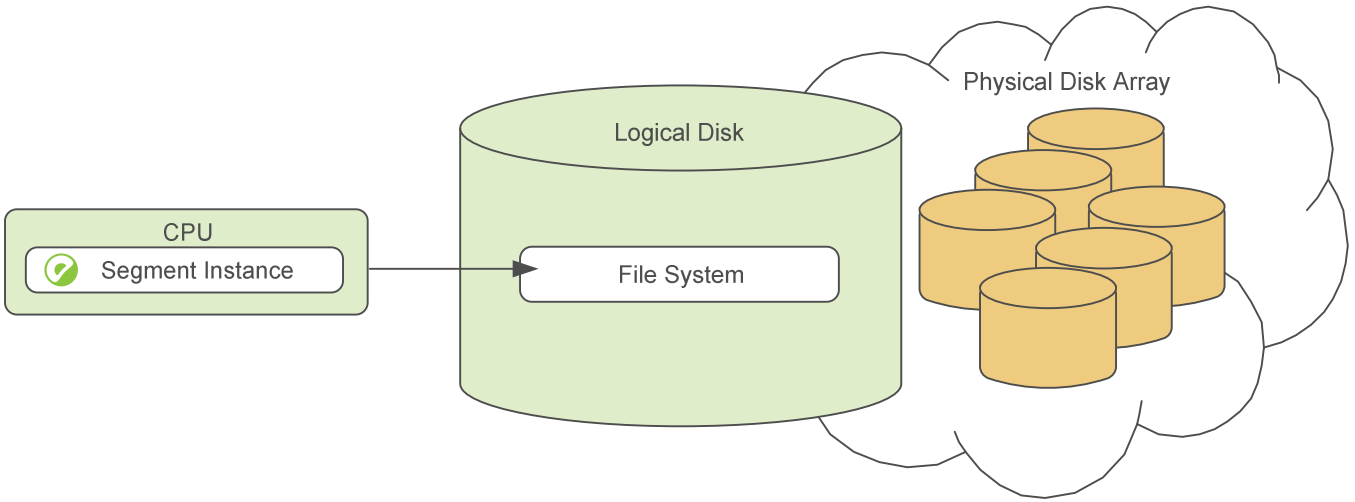
Segment主机配置示例



Greenplum

Segment主机 – 磁盘阵列

- 每个有效的CPU core对应一个主Segment实例
- 通过逻辑磁盘驱动映射到文件系统
- 逻辑磁盘驱动使用物理磁盘组(RAID)
- RAID模式的选择取决于:
  - 性能与容量的需求 (RAID-10 或 RAID-5)
  - 数据保护和磁盘容错的考虑



Greenplum

Greenplum 'White Box' 配置参考

硬件

- 15 rack units (15u)
- 5 x 3u servers (1 master + 4 segment hosts)
- 16 x 400GB SATA disks per server
- 2 SATA hardware RAID10 controllers
- 2 mirrored OS drives on separate controller
- 64 GB total RAM on segment hosts

存储能力

- 24 TB raw capacity
- 6 TB usable primary data capacity on segment hosts
- Mirroring enabled

基准性能

- 3.2GB/Second （综合的系统磁盘读写速度）

Greenplum

Sun DW设备规格

- S1002, S1004, S1010, S1020, S1040 基本规格，供客户选择
- Master 服务器(X4200) （与ETL服务器分离）
- Segment 服务器 (X4500)
- ZFS文件系统配置

Greenplum

Sun DW设备 – 2节点, 4节点, 10节点

S1002

- 1 Sun Fire X4200
- 2 Sun Fire X4500s
- 1 Sun 1000-42 Rack

S1004

- 1 Sun Fire X4200
- 4 Sun Fire X4500s
- 1 Sun 1000-42 Rack

S1010

- 1 Sun Fire X4200
- 10 Sun Fire X4500s
- 2 Sun 1000-42 Racks

Greenplum

Sun DW设备的配置

型号	S1002	S1004	S1010
容量	10 TB (20 TB with RAIDZ)	20 TB (40 TB with RAIDZ)	50 TB (100 TB with RAIDZ)
机架单位	12	20	44
读写速度	3.6 GB/sec	7.2 GB/sec	18 GB/sec
内存	32 GB	64 GB	160 GB
CPU Cores	8	16	40

Greenplum

Master主机规格(X4200)

- CPU: **2颗双核AMD Opteron处理器**
- 内存: **16 GB**
- 硬盘: **4块146GB SAS磁盘**
- 操作系统: **Solaris 10, 64位**
- 文件系统: **Solaris ZFS**
- RAID: **软件级RAID1，使用ZFS**
- 网络: **6个千兆网络接口**

Greenplum

Segment 主机规格(X4500)

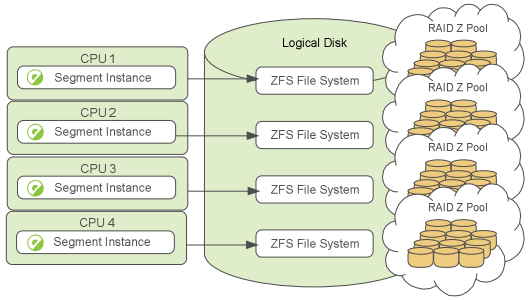
- CPU: **2颗双核AMD Opteron处理器**
- 内存: **16 GB**
- 硬盘: **48块500GB SATA II内置磁盘**
- 操作系统: **Solaris 10, 64位**
- 文件系统: **Solaris ZFS**
- RAID: **软件级RAID-Z，使用ZFS**
- 网络: **6个千兆网络接口**

Greenplum

ZFS磁盘配置

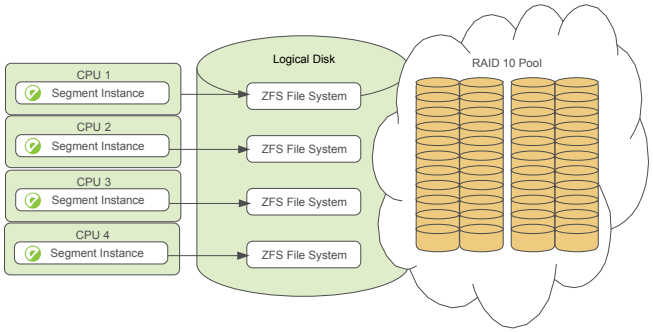
考虑存储容量的磁盘配置

- 4个RAID-Z存储池， 11块磁盘为一组 （RAID 5）
- 牺牲4块磁盘作为保护
- 读写速度大约600-800 MB/s



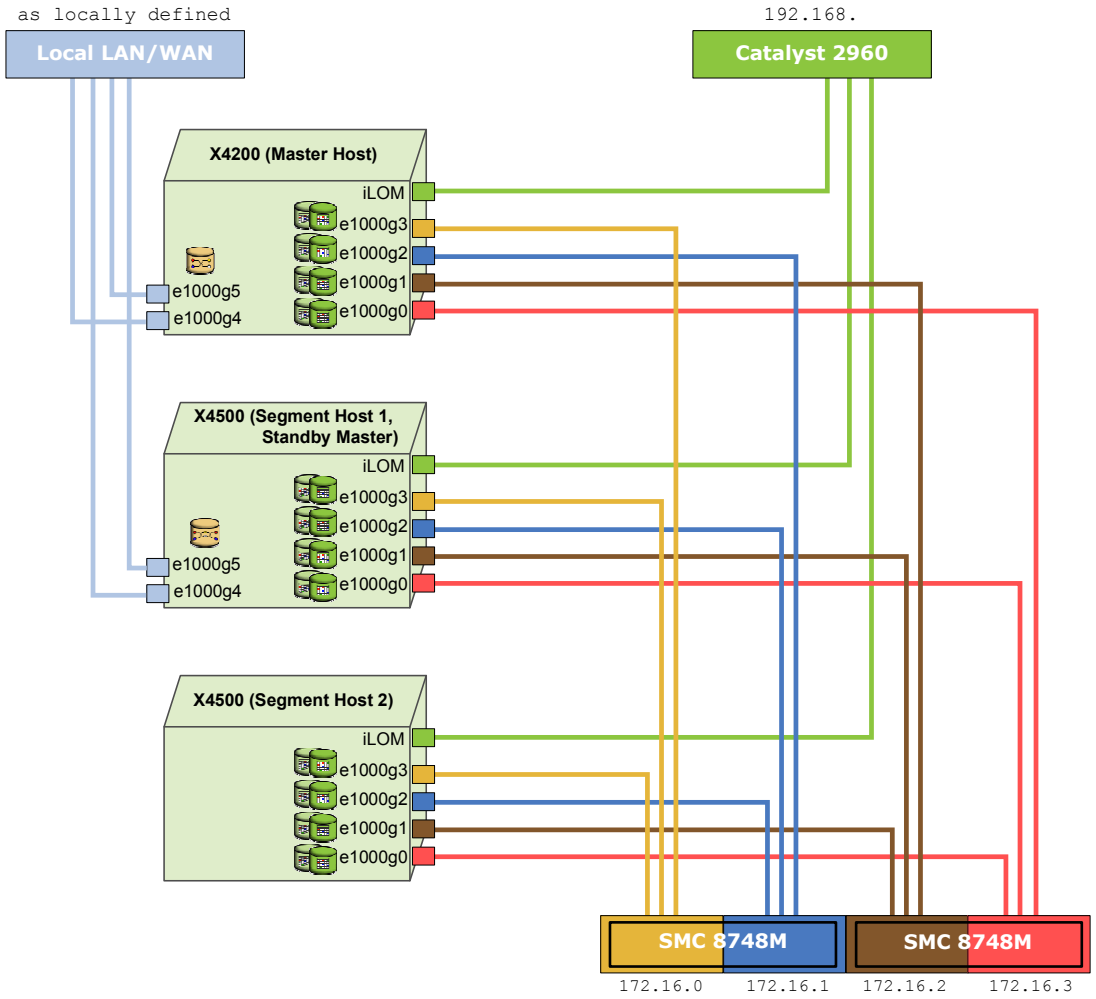
考虑高性能的磁盘配置

- 一个带镜像的存储池，以22块磁盘作为镜像组（RAID 10）
- 牺牲22块磁盘作为镜像保护
- 读写速度大约1600-1800 MB/s



Greenplum

S1002网络配置



Greenplum

其它系统准备工作

- OS内核调优
  - Linux
  - Solaris
- 硬件验证测试
- 估算存储容量

Linux系统的OS内核调优

- 共享内存
  - kernel.shmmax = 500000000
  - kernel.shmmni = 4096
  - kernel.shmall = 4000000000
  - kernel.sem = 250 64000 100 512
  - vm.overcommit_memory=2
- 网络
  - net.ipv4.tcp_tw_recycle=1
  - net.ipv4.tcp_max_syn_backlog=4096
  - net.core.netdev_max_backlog=10000
- 用户限制
  - 最大用户进程（至少131072）
  - 打开文档（至少65536）
  - 编辑/etc/security/limits.conf来增加对用户的限制

Greenplum

Solaris系统的OS内核调优

- 网络 (/etc/rc2.d)
  - tcp_conn_req_max_q=4096
  - tcp_conn_req_max_q0=4096
  - tcp_largest_anon_port=65535
  - tcp_smallest_anon_port=4096
  - tcp_time_wait_interval=1000
- 打开文件描述符 (/etc/system)
  - rlim_fd_max=262144
  - rlim_fd_cur=65536
- 共享内存(/etc/system)
  - shmsys:shminfo_shmmax=0x2000000
  - semsys:seminfo_semmni=1024
- 配置后重启...

Greenplum

硬件的验证和测试

- 测试硬盘的读写速度 (gpcheckperf命令)
- 测试硬盘存储的带宽(gpcheckperf命令)
- 测试服务器间的网络传输速度 (gpchecknet/gpcheckperf命令)
- 验证操作系统的环境设置 (gpcheckos)
- 压力测试 (bonnie++)

Greenplum

估算存储容量

从总的磁盘空间中减去…

- RAID类型/镜像
- Greenplum数据库的segment镜像
- 文件系统的损益
- 不要完全使用磁盘容量（推荐数据量小于磁盘容量的70%）
- 原始数据大小+数据库存储损耗
- 系统元数据（每个segment实例大约20MB）
- WAL （每个segment实例大约1088MB）
- Greenplum数据库日志文件（每个segment实例大约10MB，需要维护）

Greenplum

第2课练习

系统验证命令

第3课

安装和初始化

- 安装概述
- 环境设置
- 系统初始化
- 生产环境的Greenplum集群
- 虚拟的Greenplum 集群
- 多系统的Greenplum集群

安装概述

1. 在Master上运行Greenplum数据库安装程序
2. 在Master上设置Greenplum的路径(`greenplum_path.sh`)
3. 在每台主机上设置OS调优环境
4. 在每台主机上创建Greenplum的超级用户(`gpadmin`)
5. 在root和gpadmin用户下交换SSH密钥(`gpssh-exkeys`)
6. 在每台主机上同步系统时钟(`ntp`)
7. 创建master和segment上的数据目录位置
8. 创建Greenplum安装压缩包，复制到各segment主机并解压缩(`gpscp` and `gpssh`)
9. 使用期望的集群配置初始化Greenplum数据库系统 (`gpinitsystem`)
10. 设定附加的环境变量 (Master和热备机的`.bashrc`文件)

Greenplum

Greenplum数据库系统初始化

1. 创建主机列表文件（所有segment主机名称）
2. 编辑系统配置文件 (gp_init_config)
3. 在master主机上运行gpinitsystem 命令

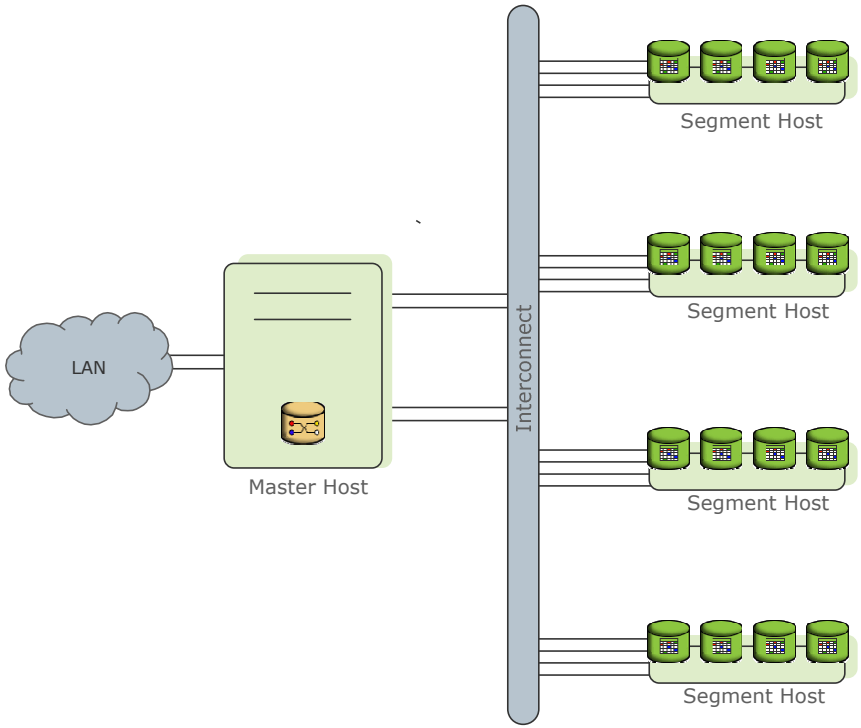
例如:

```
gpinitsystem -c gp_init_config
```

Greenplum

生产环境的Greenplum集群

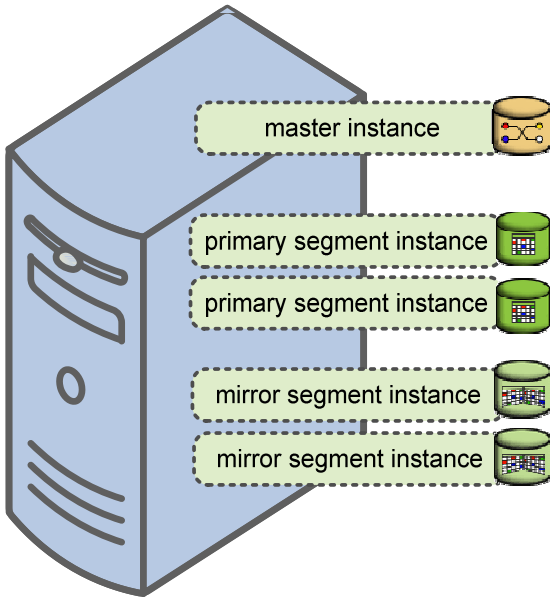
- 1个master主机和2个以上的segment主机
- 每个CPU对应一个主segment实例
- 每个Segment配置多个网络接口（推荐）



Greenplum

虚拟化(Demo) Greenplum集群

- Master 和所有segment实例在一台机器上
- 只作为演示或开发的用途
- 非生产的Greenplum架构，不能测试性能

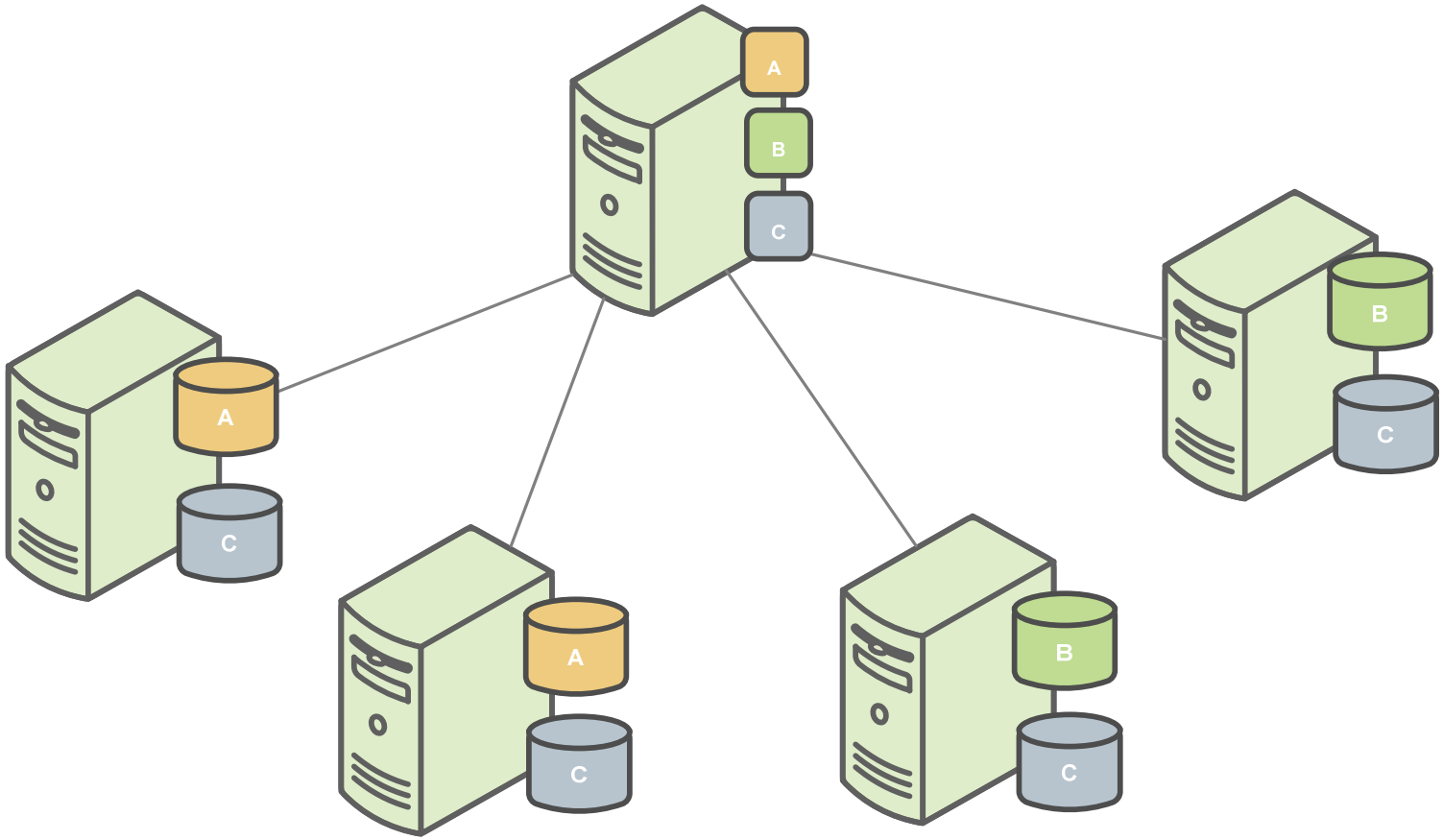


Greenplum

多系统的Greenplum集群

- 两个或以上的独立Greenplum数据库系统共享同样的硬件资源
- 每个Greenplum系统使用不同的Greenplum管理用户（即超级用户）
- 创建不同的数据目录位置
- 确保指定端口不发生冲突

多系统集群的例子



Greenplum

Greenplum升级方式（小版本更新）

- 在每台主机上安装新的软件包
- 关闭Greenplum数据库(`gpstop`)
- 更新链接路径到新的安装包上
- 重新启动Greenplum数据库(`gpstart`)

第3课练习

Greenplum数据库的安装和初始化

第4课

客户端应用程序

- 通过PSQL连接
- pgAdmin III

PSQL客户端连接

- 通过Master连接数据库系统
- 连接信息
  - 数据库名称(-d | PGDATABASE)
  - master主机名称 (-h | PGHOST)
  - master 端口(-p | PGPORT)
  - 用户名 (-U | PGUSER)
- 第一次连接
  - 默认的`template1`数据库
  - 默认的superuser (`gpadmin`)

Greenplum

执行SQL语句的方式

- 交互方式

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

- 非交互方式 (单行命令)

```
psql mydatabase -c “SELECT * FROM table;”
```

- 非交互方式 (多行命令)

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

- 每个SQL语句的结束需要使用分号(;) 来表示

Greenplum

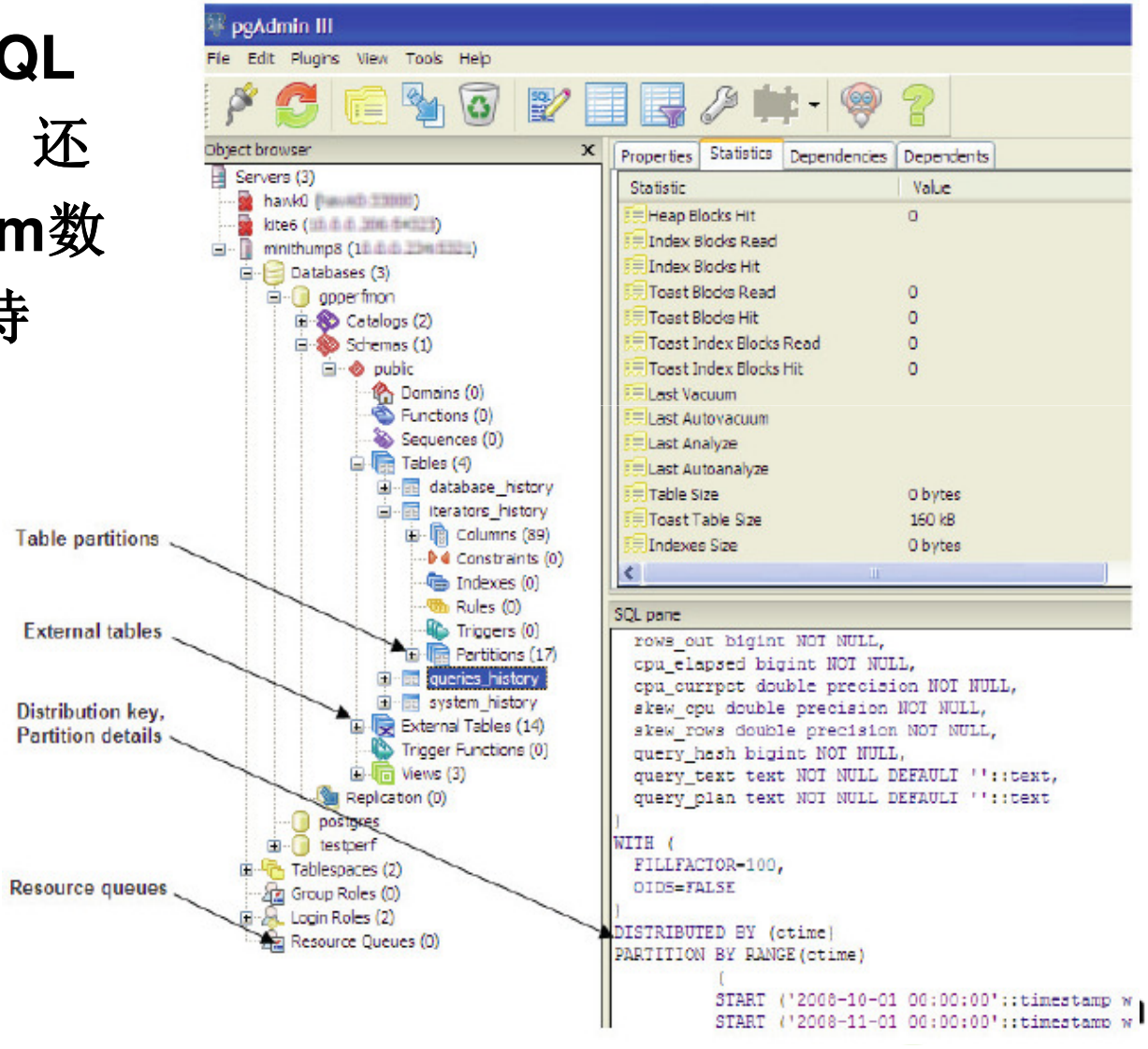
PSQL基本使用命令

- `\?` **(psql查询命令帮助)**
- `\h` **(SQL命令语法帮助)**
- `\dt` **(显示表)**
- `\dtS` **(显示系统表)**
- `\dg` **or** `\du` **(显示角色)**
- `\l` **(显示数据库)**
- `\c` *db_name* **(连接到这个数据库)**
- `\q` **(退出psql)**

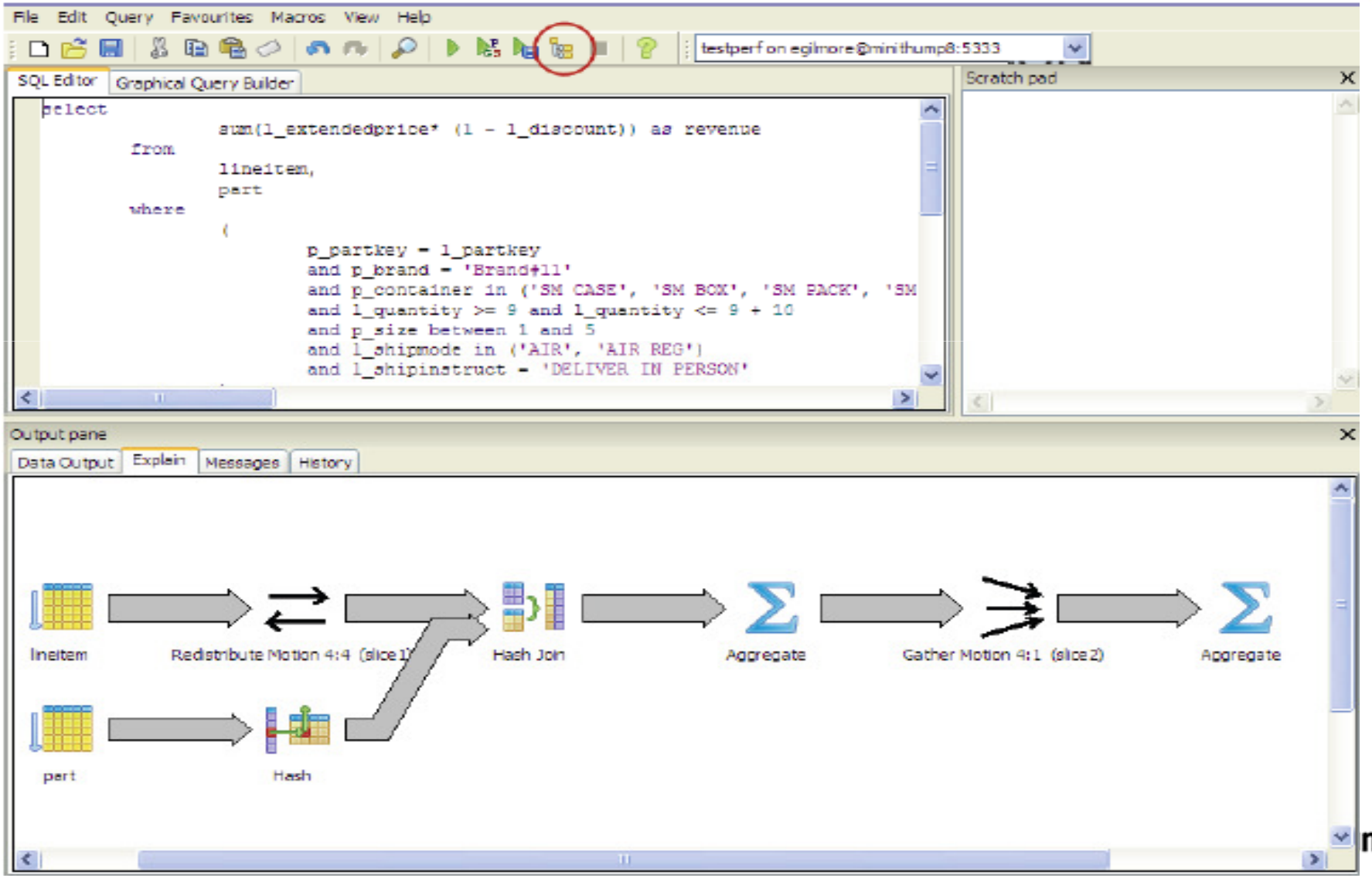
Greenplum

图形界面的客户端 – pgAdmin III

- **除了支持PostgreSQL数据库的查询使用，还增加了对Greenplum数据库特殊功能的支持**
  - 外部表
  - 表分区
  - 资源队列
  - 图形化的SQL解析
  - 数据库配置参数



使用pgAdmin III查看图形化的查询计划



第4课练习

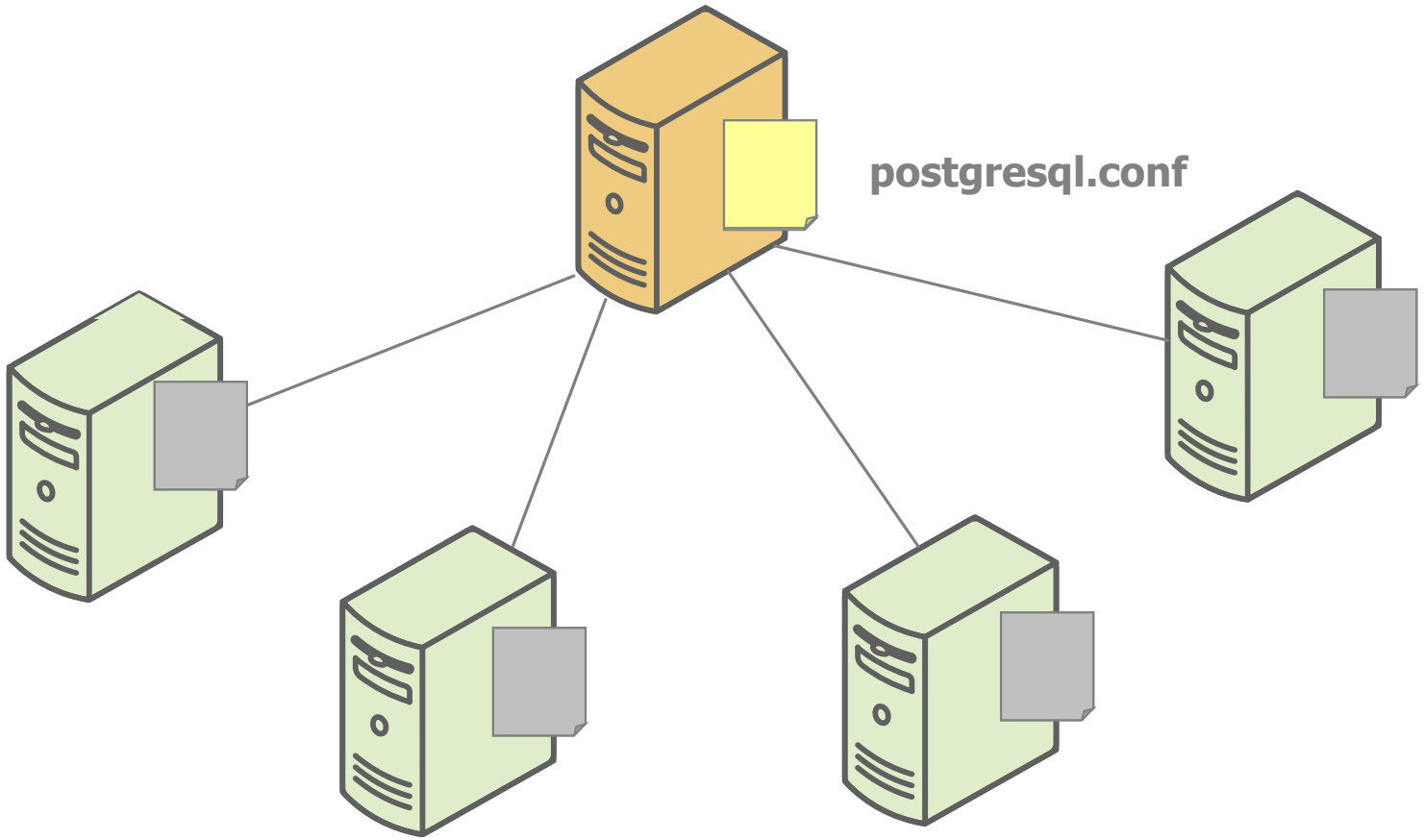
使用PSQL

Module 5

服务器配置

- 本地、全局和仅用于master的参数
- 配置文件postgresql.conf
- 设定配置参数
- 查看参数的设置
- 配置参数的类别
- 基于主机的认证文件 (pg_hba.conf)

本地，全局和仅用于Master的参数



Greenplum

postgresql.conf文件

- 位于master或segment实例的数据目录下
- 用于设定系统级的配置参数
- 使用默认设置的参数用注释方式（#）
- 需要重启系统使修改的参数生效（或使用 `gpstop -u`再加载）

Greenplum

设定配置参数

- 某些参数只能在系统启动时设定
- 运行时的参数可以根据用户级，数据库级或对话级别的设定 （用户可操作）
- 某些参数需要超级用户的权限
- 某些参数是只读方式

Greenplum

设定配置参数

用户操作的全局和仅用户master的参数

- **系统级别（编辑master的postgresql.conf文件）**
  例如：`log_min_messages = DEBUG1`
- **数据库级别（使用ALTER DATABASE命令）**
  例如：`ALTER DATABASE names SET search_path TO baby, public, pg_catalog;`
- **用户/群组级别（使用ALTER ROLE命令）**
  例如:`ALTER ROLE lab1 SET search_path TO baby, public, pg_catalog;`
- **客户端对话级别（使用SET命令）**
  例如:`SET search_path TO  baby, public, pg_catalog;`

本地参数

- **Segment级别（编辑segment的postgresql.conf文件）**

Greenplum

查看参数的设置

- **查看某个特定参数设置**

  例如: `SHOW search_path;`

- **查看所有参数设置**

  例如: `SHOW ALL;`

Greenplum

配置参数类别

- 连接和认证
- 资源消耗
- 预写日志(WAL)
- 查询计划（只和master相关）
- 日志
- 运行时统计
- Autovacuum
- Greenplum特别内容 (gp_* 前缀)

Greenplum

配置基于主机的认证

- **客户端认证**
  - 从这个客户端主机连接系统是否允许？
  - 用户连接到某个数据库是否有权限？
- **pg_hba.conf文件**
  - 控制通过主机地址、数据库和/或数据库用户连接的认证方式
  - 位于master或segment实例的数据目录下
  - 系统初始化时设定默认配置

默认Segment主机的pg_hba.conf

- 只允许从master主机进行远程连接
- (可选项) 也允许热备的master主机远程连接
- 例子:

```
# TYPE    DATABASE    USER    CIDR-ADDRESS     METHOD
  local   all         all                      trust
  host    all         all     127.0.0.1/32     trust
  host    all         all     ::1/128          trust
  host    all         all     10.0.0.206/32    trust
```

Greenplum

默认Master主机的pg_hba.conf

- 允许Greenplum超级用户本地连接
- 不允许远程连接
- 例子:

```
# TYPE    DATABASE    USER    CIDR-ADDRESS    METHOD
  local     all      gpadmin                 ident sameuser
  local     all      all                     ident sameuser
  host      all      gpadmin     127.0.0.0/0     trust
```

Greenplum

第5课练习

5A: 设置服务器配置参数
5B: 配置基于主机的认证文件

第6课

数据定义语言(DDL)

- 数据库
- **Schemas**
- 表
- 数据类型
- 约束条件
- 其它数据库对象

关于数据库

- 一个Greenplum数据库系统可以有多个数据库
- 客户端每次只连接一个数据库
- 默认数据库模板: `template1`
- 其它数据库模板: `template0, postgres`

数据库相关SQL命令

- **创建:** `CREATE DATABASE` **or** `createdb`
- **删除:** `DROP DATABASE` **or** `dropdb`
- **编辑:** `ALTER DATABASE`
  - 修改名称
  - 指定新的所有者
  - 设定配置参数

Greenplum

PSQL连接数据库的提示

- **PSQL 界面会显示当前连接的数据库**
  例子: `template1=#` **(超级用户)**
  `names=>` **(非超级用户)**
- **显示所有数据库列表:**
  `\l`
- **连接到另一个数据库:**
  `\c` *`db_name`*
- **使用PGDATABASE环境变量来设置默认的数据库**

Greenplum

关于Schemas

- Schemas是在一个数据库中将对象逻辑上组织的方式（表，函数等）
- 使用带有schema的“完整的”名称来读取数据库对象
  例如: `myschema.mytable`
- 对于只有对象名称的内容，通过schema的搜索路径来决定默认的schema (`search_path`)
- 每个数据库有名称为public的schema
- 其它系统级别的schemas: `pg_catalog`, `information_schema`, `pg_toast`, `pg_bitmapindex`

Greenplum

Schema相关SQL命令

- **创建:** `CREATE SCHEMA`
- **删除:** `DROP SCHEMA`
- **编辑:** `ALTER SCHEMA`
  - 修改名称
  - 指定新的所有者

PSQL中Schema相关提示

- **查看当前的schema:**

```
SELECT current_schema();
```

- **查看数据库中所有schema列表:**

```
\dn
```

- **查看schema的搜索路径:**

```
SHOW search_path;
```

- **设置某个数据库的搜索路径:**

```
ALTER DATABASE SET search_path TO
myschema, public, pg_catalog;
```

关于表

- Greenplum数据库的所有表都会根据分布键分布到每个Segment实例上
- 3.3版本以上可以通过update table修改表的分布键，在表已经创建后，之前的版本没有此功能
- 某些表功能目前在Greenplum数据库中不支持
  - 外键约束（Foreign key)
  - 唯一约束的限制 (Unique constraints)

关于表分布键

- 使用一列或多列内容来分隔数据到各segment上
- 为确保数据分布均衡，最好是唯一的
- 在CREATE TABLE是使用DISTRIBUTED BY语句来定义分布键

```
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);
```

- 如果表中没有唯一值的列，可以定义DISTRIBUTED RANDOMLY作为分布键
- 如果没有明确定义，默认表的主键或表的第一列作为分布键

Greenplum

关于数据类型

- 描述表列所包含数据的类型
- Greenplum支持与PostgreSQL相同的数据类型
- Greenplum同样支持SQL标准定义的所有数据类型
- Greenplum分布键设置类型
  - 不能使用几何数据类型（如box, circle, line, 等）
  - 不能使用用户自定义数据类型

关于表和列的约束条件

- CHECK 表或列的约束
- NOT NULL 列约束条件
- UNIQUE 列约束条件
  - 每个表只能有一个
  - Unique定义的列会作为表分布键
  - 如果表设置了主键，则不允许有Unique约束
- PRIMARY KEY 表约束条件
  - 默认会作为表分布键
- FOREIGN KEY 约束目前不支持

表相关的SQL命令

- **创建:** CREATE TABLE
  - 增加DISTRIBUTED BY或DISTRIBUTED RANDOMLY从句
  - 某些语法不支持
- **编辑:** ALTER TABLE
- **删除:** DROP TABLE

关于外部表和网络表

- 两种类型
  - 外部表
    - 建立Greenplum 文件服务器(gpfdist)
    - 基于文件(file://)
  - 网络表
    - 基于URL (http://)
    - 基于输出内容 (OS命令，脚本)
- 可用于ETL和数据加载
- 数据加载中有隔离错误行的功能
- 数据通过Segment来读取（并行方式）
- 只读形式

Greenplum

外部表相关SQL命令

- **创建:** CREATE EXTERNAL [WEB] TABLE
  - LOCATION**从句**
  - SEGMENT REJECT LIMIT**从句**
  - FORMAT**从句**
- **删除:** DROP EXTERNAL [WEB] TABLE

表相关提示

PSQL 提示

- **列出数据库的所有表:**

  ```
  \dt
  ```

- **查看表的结构:**

  ```
  \d+ table_name
  ```

- **列出所有系统目录表:**

  ```
  \dtS
  ```

- **列出数据库的外部表:**

  ```
  \dx
  ```

Greenplum管理提示

- **查看表的分布键:**

  ```
  gpskew -t table_name -c
  ```

Greenplum

其它数据库对象

- 视图
- 索引
- 函数和运算符
- 序列
- 触发器（Triggers）
- 表空间

关于视图(Views)

- 将频繁使用的查询定义为视图
- 视图在Greenplum数据库中不是实体
- 视图相关SQL命令：
  - CREATE VIEW
  - DROP VIEW
- PSQL提示：
  - 显示所有的视图：\dv
  - 查看一个视图的定义：\d+ *view_name*
- 实例：

```
CREATE VIEW topten
   AS SELECT name, rank, gender, year
   FROM names, rank
   WHERE rank < '11' AND names.id=rank.id;

SELECT * FROM topten ORDER BY year, rank;
```

Greenplum

关于索引(Indexes)

- 索引在Greenplum数据库中保守使用
- 索引在查询运行中并不是都有帮助
- 需要测试索引是否能有效提高性能
- 不使用的情况下应删除索引
- PRIMARY KEY索引会自动创建
- Unique索引只允许在Greenplum分布键所在的列上

关于索引(continued)

- **索引的类型：**
  - B-tree
  - **Bitmap***
  - R-tree
  - Hash
  - GiST
- **索引相关的SQL命令：**
  - `CREATE INDEX`
  - `ALTER INDEX`
  - `DROP INDEX`
  - `*CLUSTER`
  - `REINDEX`
- **PSQL提示：**
  - **显示所有的索引：** `\di`
  - **查看一个索引的定义：** `\d+ index_name`

Greenplum

关于位图索引

- 位图索引对于数据仓库应用是理想方式
- 位图索引有较小的存储容量
- 索引数据用位图方式存储
- 主要应用场景：
  - 低基数的列
  - 小于5000个固有值的列
  - 不会频繁更新的表
  - 查询包含多个WHERE条件
- 不适合的场景：
  - OLTP应用
  - 更新频率高的列
  - 唯一限制或高基数的列

Greenplum

位图索引实例

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 male   - 10111010
bitmap for female - 01000101
```

Greenplum

关于序列(Sequences)

- **Greenplum Master序列生成器进程（`seqserver`）**
- **序列相关的SQL命令：**
  - `CREATE SEQUENCE`
  - `ALTER SEQUENCE`
  - `DROP SEQUENCE`
- **序列有关函数的限制：**
  - **不支持`lastval`和`currval`**
  - **查询中更新数据不能使用`setval`**
  - **如果Mirrors启用时，`nextval`在更新和删除命令中是不允许的**
  - **`nextval`在某些查询中会获取一批数值**
- **PSQL提示：**
  - **显示所有的序列：`\ds`**
  - **查看一个序列的定义：`\d+` *`sequence_name`***

Greenplum

序列实例

- 创建一个名为“myseq”的序列：

```
CREATE SEQUENCE myseq START 101;
```

- 插入一行到表中并获得下一个序列值：

```
INSERT INTO distributors
        VALUES (nextval('myseq'), 'acme');
```

- 在master上重设序列计数的值：

```
SELECT setval('myseq', 201);
```

- Greenplum数据库不允许的 (在segments上设置序列值)：

```
INSERT INTO product
        VALUES (setval('myseq', 201), 'gizmo');
```

Greenplum

关于内置函数和运算符

- Greenplum支持所有种类PostgreSQL提供的内置函数和运算符
- 完全支持不可变函数
- 对于稳定的和不稳定的可变函数使用有限制条件

稳定和不稳定的函数

- 支持仅在Master上运行的函数命令（没有FROM语句）：
  - `SELECT setval('myseq', 201);`
  - `SELECT foo();`
- 不支持函数命令运行在segment级别且函数中包含SQL或更改数据库的命令：
  - `SELECT * FROM foo();`
- 不支持函数用于更新或删除命令且Mirrors是启用的：
  - `UPDATE mytable SET id = nextval(myseq);`

Greenplum

关于用户自定义函数

- Greenplum数据库支持所有基于PostgreSQL的服务器扩展功能
- 对于稳定和不稳定的函数使用有限制条件
- 当设置新的函数时：
  - 声明适当的变动级别（不稳定，稳定，不可变）
  - 将共享库文件放置在每台主机相同的库文件路径下（master，segments，mirrors）
  - 任何函数使用自定义的数据类型、运算符或集合时都应该是不可变的

Greenplum

关于触发器(Triggers)

- 触发器——数据库在某些特定操作执行时，会自动触发并执行一些特殊函数
- Greenplum数据库目前不支持触发器

关于表空间(Tablespaces)

- 表空间——在文件系统中可设置的存储单元，保存数据库、表、索引等数据
- Greenplum数据库目前不可用

第6课练习

6A: 数据库和Schemas
6B: 表和分布键
6C: 视图，索引，序列

第7课

角色，权限和资源队列

- 关于角色和权限
- 数据库角色（用户）
- 角色成员（用户组）
- 数据库对象的权限
- 工作量管理（基于角色的资源队列）

关于角色和权限

- 用户，用户组（或两者都是）
- 与操作系统的用户和群组不相关
- 确定权限级别的属性
- 每个系统有一个默认的超级用户
- 角色会被授予数据库对象的存取权限
- 角色可以是其它角色的成员
- 角色是在系统级别被定义

角色（数据库用户）

- 一个用户账号就是一个可以登录的角色
- 一个新定义角色的默认属性包括：
  - NOSUPERUSER
  - NOCREATEDB
  - NOCREATEROLE
  - INHERIT
  - NOLOGIN (必须明确将LOGIN属性赋予用户级别的角色）
- 使用CREATE ROLE添加一个新角色
- 使用ALTER ROLE 对已有角色进行权限属性修改
- 实例：

```
CREATE ROLE john WITH LOGIN;
ALTER ROLE john WITH CREATEDB;
CREATE USER john;
```

Greenplum

角色成员（用户组）

- 一个角色可以是其它角色的成员
- 成员可继承用户组的对象权限
- 允许在同一个地方设置对象权限
- 角色权限不继承，可使用SET ROLE连接到用户组来使用其权限
- 用户组一般没有登录的权限
- 使用GRANT命令来授予成员资格
- 使用REVOKE命令将成员从用户组删除
- 实例：

```
CREATE ROLE admin CREATEROLE CREATEDB;
GRANT admin TO john, sally;
REVOKE admin FROM bob;
SET ROLE admin;
```

关于数据库对象权限

- 每个数据库对象都有一个“所有者”角色，拥有这个对象的所有权限
- 其它角色必须被授予相关的权限
- 授予权限给PUBLIC，意味着将权限给所有角色
- 使用GRANT命令授予对象权限，REVOKE命令取消权限
- 使用DROP OWNED和REASSIGN OWNED来取消对象当前的所有权
- 实例：

```
GRANT ALL ON DATABASE mydatabase TO admin WITH GRANT
  OPTION;
GRANT SELECT ON TABLE mytable TO PUBLIC;
REVOKE INSERT UPDATE ON TABLE mytable FROM sally;
REASSIGN OWNED BY sally TO bob;
DROP OWNED BY visitor;
```

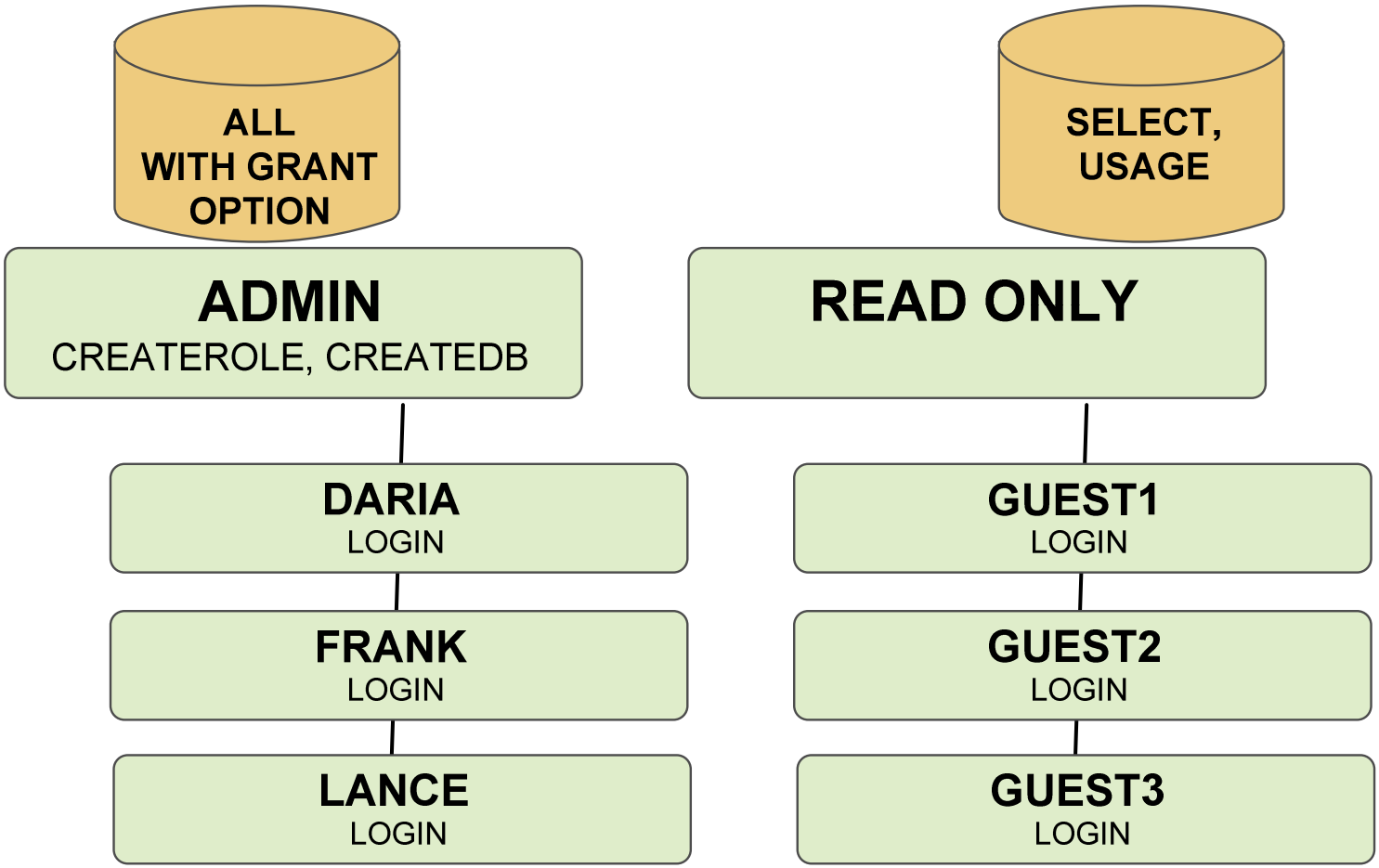
Greenplum

不同对象的可分配权限

- **表，视图，序列**
  - SELECT
  - INSERT
  - UPDATE
  - DELETE
  - RULE
  - ALL
- **数据库**
  - CREATE
  - TEMPORARY
  - ALL
- **函数**
  - EXECUTE
- **程序语言**
  - USAGE
- **Schemas**
  - **CREATE**
  - **USAGE**
  - **ALL**

Greenplum

数据库角色结构的实例



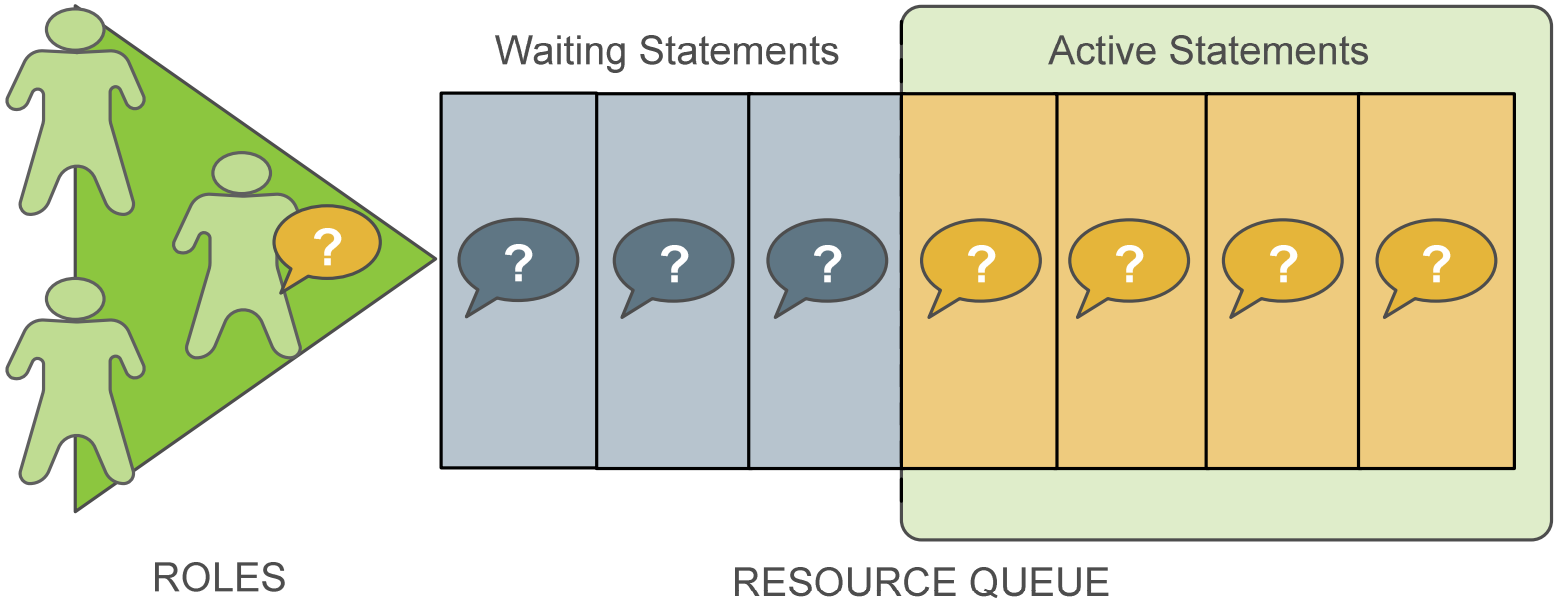
Greenplum

Greenplum的工作量管理

- 工作量管理是指什么?
- 创建资源队列
- 分配角色到资源队列
- 资源队列在运行中的估算
- 资源队列的配置参数
- 查看资源队列的状态

工作量管理是指什么？

- **用于限制当前查询的数量**
- **出于保护系统资源（CPU，磁盘读写，存储）的过载**



Greenplum

创建资源队列

- **SQL命令**
  - CREATE RESOURCE QUEUE
  - ALTER RESOURCE QUEUE
  - DROP RESOURCE QUEUE
- **资源队列的限制条件**
  - **活跃查询限制**

    例如：CREATE RESOURCE QUEUE adhoc ACTIVE THRESHOLD 10 IGNORE THRESHOLD 1000.0;
  - **使用资源限制**

    例如：CREATE RESOURCE QUEUE batch1 COST THRESHOLD 1000000.0 NOOVERCOMMIT;

    CREATE RESOURCE QUEUE batch1 COST THRESHOLD 1e+6;

Greenplum

分配角色到资源队列

- 必须在用户级别分配（用户组级别忽略）
- 超级用户不受资源队列限制条件的约束
- SQL命令
  - **ALTER ROLE**

    ```
    ALTER ROLE role_name RESOURCE QUEUE queue_name;
    ```

  - **CREATE ROLE**

    ```
    CREATE ROLE role_name WITH LOGIN RESOURCE QUEUE
    queue_name;
    ```

Greenplum

资源队列在运行中的估算

- 资源队列进行互相独立的估算
- 超级用户（和没有分配角色）不计算在内
- 查询估算使用先进先出原则
- 如果一个查询导致队列超出他的限制条件：
  - 该查询必须等待队列资源有空闲才执行，或者
  - 该查询必须等待队列没有其它查询才执行(`cost overcommit`), 或者
  - 该查询不会被执行(`cost no overcommit`)
- 与估算有关的SQL命令
  - `SELECT, SELECT INTO, CREATE TABLE AS SELECT, DECLARE CURSOR`
  - (可选的) `INSERT, UPDATE, DELETE`

Greenplum

资源队列的配置参数

- resource_scheduler = on
- max_resource_queues = 8
- max_resource_portals_per_transaction = 64
- resource_select_only = on
- stats_queue_level = off

Greenplum

查看资源队列的状态

- 查询系统表：
  - pg_resqueue (资源队列和属性)
  - pg_roles (资源队列的分配角色)
  - pg_locks (队列存在等待的状态)
  - pg_stat_activity (执行和等待的查询进程信息）
- 队列状态和统计的视图
  - pg_resqueue_status (队列限制，执行和等待查询的数量)
  - pg_stats_resqueue (队列统计和时间上的性能)

第7课练习

7A: 角色和权限
7B: 基于角色的资源队列

第8课

表分区

- 表分区的概念
- 创建分区表
- 维护分区表

什么是表分区?

- 将一张大表逻辑性地分成多个部分
- 提高对于特定类型数据的查询速度和性能
- 也可以更方便数据库的维护和更新
- 两种类型:
  - *Range分区 (日期范围或数字范围)*
  - *List 分区*
- **Greenplum**中的表分区在使用中具有母表的继承性，并通过Check条件指定相应的子表
- 分区的子表依然根据分布策略分布在各**segment**上

Greenplum

什么时候需要分区?

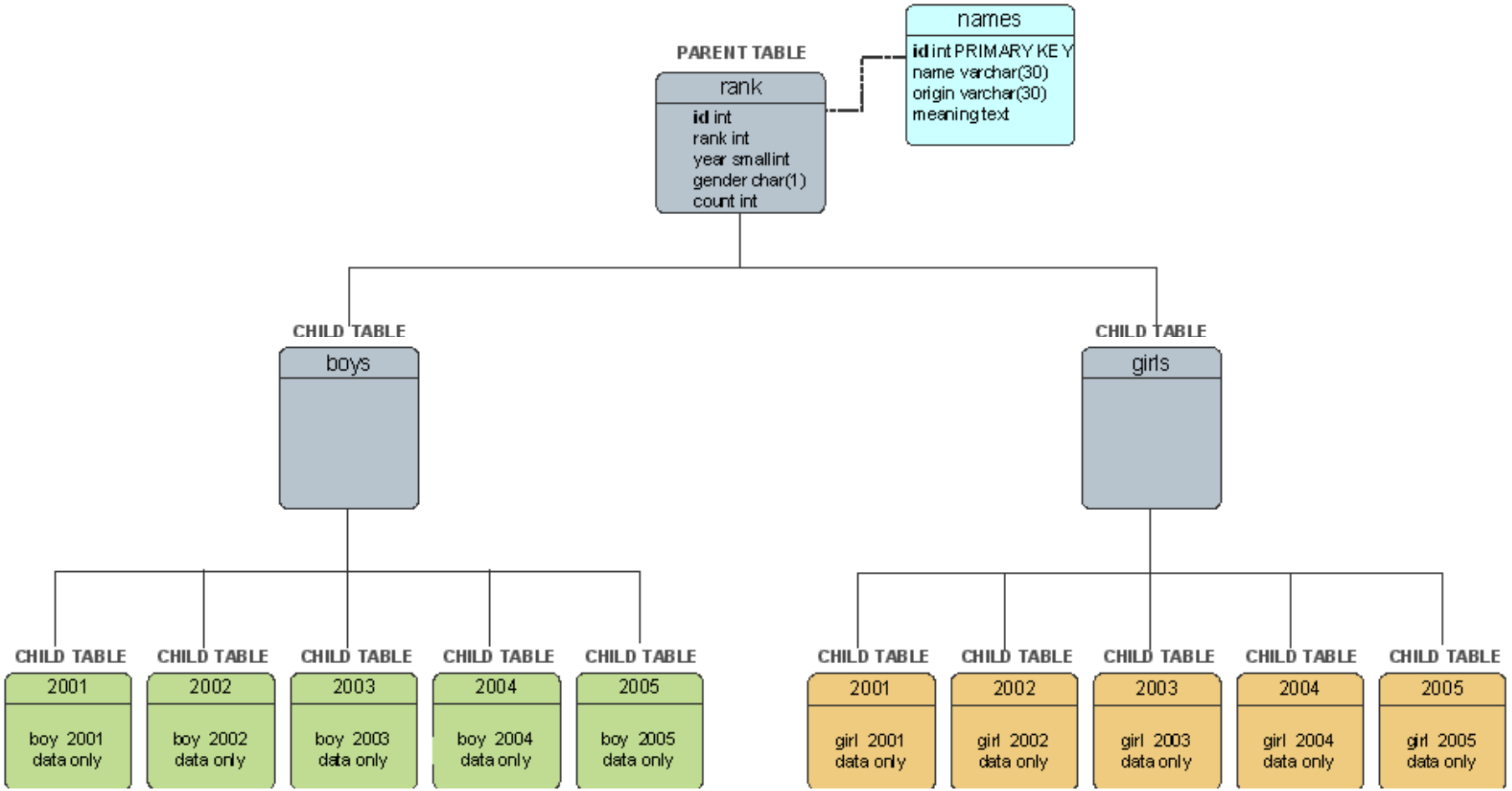
- 一张大表（事实表）
- 性能体验不令人满意
- 查询描述中具有明显的存取模式(例如： `WHERE date = '12/13/2007'`)
- 需要维护“滚动窗口”的数据
- 数据能通过一些定义标准分成均衡的几部分

Greenplum

理解表分区

- 创建子表
  - 从母表上继承
  - 使用CHECK表约束条件来定义
- 在母表上定义重写规则
- 只将数据加载到每个子表上
- 查询计划会决定扫描哪些表分区，以此获得满意的查询效果
- 在运行中—向通常一样查询母表

实验环境的分区的表实例



Greenplum

创建分区表

- 表分区只能在创建表时通过CREATE TABLE命令来实现
- 定义日期范围的表分区
- 定义数值范围的表分区
- 定义列表的表分区
- 定义多级的分区

创建日期范围的分区

- 定义日期范围的表分区

```
CREATE TABLE sales (id int, date date, amt
  decimal(10,2))
DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
( START (date '2008-01-01') INCLUSIVE
END (date '2009-01-01') EXCLUSIVE
EVERY (INTERVAL '1 day') );
```

Greenplum

创建数字范围的分区

- 定义数字范围的表分区

```
CREATE TABLE rank (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 );
```

创建列表的分区方式

- 定义列表的表分区

```
CREATE TABLE rank (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 );
```

创建多级别的分区表

- 定义多级分区表

```
CREATE TABLE sales (trans_id int, date date,
  amount decimal(9,2), region text)
DISTRIBUTED BY (trans_id)
PARTITION BY RANGE (date)
SUBPARTITION BY LIST (region)
SUBPARTITION TEMPLATE
( SUBPARTITION usa VALUES ('usa'), SUBPARTITION
  asia VALUES ('asia'), SUBPARTITION europe
  VALUES ('europe') )
( START (date '2008-01-01') INCLUSIVE
END (date '2009-01-01') EXCLUSIVE
EVERY (INTERVAL '1 month') );
```

Greenplum

维护分区表

- 分区表维护使用ALTER TABLE命令
- 添加新的分区表
- 对分区表重新命名
- 删除分区表
- 清空分区表
- 拆分分区表

添加分区表

- 使用ALTER TABLE 命令对已存在的分区添加一个新的分区表

```
ALTER TABLE sales ADD PARTITION
   START (date '2009-02-01') INCLUSIVE
   END (date '2009-03-01') EXCLUSIVE;
```

- 如果原始的分区设计包括使用subpartition template创建的子分区，添加的分区表也会依照模板增加相应的子分区

对分区表重新命名

- **分区表的命名规则：**

  <parentname>_<level>_prt_<partition_name>

  例如： *sales_1_prt_jan08*

- **如果没有定义分区名称，系统会用数字来赋予给自动生成的分区表：**

  例如： *sales_1_prt_1*

- **如果重新命名了母表名称，所有和它关联的子分区表中** *<parentname>* **也会相应更改**

  例如： `ALTER TABLE sales RENAME TO globalsales;`

- **更改分区表的分区名称**

  例如：
  ```
  ALTER TABLE sales RENAME PARTITION FOR ('2008-01-
  01') TO jan08;
  ```

Greenplum

删除分区表

- 如果删除的分区表包含有子分区表，子分区表和其中的数据也会同样被自动删除。
- 对于设置范围的分区表，通常从范围中清除历史分区表，相应的历史数据也会从数据仓库中清除。

  例如：`ALTER TABLE sales DROP PARTITION FOR (RANK(1));`

清空分区表

- 如果清空的分区表包含有子分区表，子分区表也会自动清空。

  例如：`ALTER TABLE sales TRUNCATE PARTITION FOR (RANK(1));`

拆分分区表

- 拆分分区表就是将已存在的一个分区表分开成两个分区表
- 只能在最低级别进行分区表拆分（即只能拆分包含数据的分区表）
- 指定的拆分值会应用于拆分的后半部分分区表
- 例如：

```
ALTER TABLE sales SPLIT PARTITION FOR ('2008-01-
  01') AT ('2008-01-16')
INTO (PARTITION jan081to15, PARTITION
  jan0816to31);
```

Greenplum

分区表的限制

- 为了能够执行主键或唯一键的限制条件，主键或唯一键所在列必须从分区设置列开始。
- 母表上创建的索引并不会作用在子分区表上，每个子分区表包含的数据必须直接创建索引。
- 对母表进行ANALYZE操作不会收集子分区表的统计信息，每个子分区表必须单独进行ANALYZE的操作。
- 对母表进行VACUUM操作也不会清理子分区表的内容，每个子分区表必须单独进行VACUUM的操作。

第8课练习

8A: 创建分区表
8B: 创建多级分区表
8C: 添加/删除/重新命名分区表

第9课

数据加载

- 外部表功能
- COPY 命令
- 数据加载的性能提示

使用外部表功能加载

- 外部表的特征
  - 数据存放在数据库外
  - 可执行SELECT, JOIN, SORT等命令，类似正规表的操作
- 外部表的优点
  - 并行方式加载
  - ETL的灵活性
  - 格式错误行的容错处理
  - 支持多种数据源
- 两种方式:
  - External Tables: 基于文件
  - Web Tables: 基于URL或指令

基于文件的外部表

- 两种访问协议
  - file
  - gpfdist
- 文件格式
  - text
  - csv
- 外部表定义的实例

```
CREATE EXTERNAL TABLE ext_expenses (name text, date date, amount
   float4, category text, description text)
LOCATION (
'file://seghost1/dbfast/external/expenses1.csv',
'file://seghost1/dbfast/external/expenses2.csv',
'file://seghost2/dbfast/external/expenses3.csv',
'file://seghost2/dbfast/external/expenses4.csv',
'file://seghost3/dbfast/external/expenses5.csv',
'file://seghost3/dbfast/external/expenses6.csv',)
FORMAT 'CSV' ( HEADER );
```

Greenplum

并行的文件分布程序(gpfdist)

- 确保完全的并行，提供最佳的性能
- 能够运行在Greenplum阵列之外的服务器上
- C程序 – 使用HTTP协议
- 每个gpfdist程序提供约200 MB/s的数据分配速率
- 配置参数: `gp_external_max_segs`
- 启动程序命令实例:

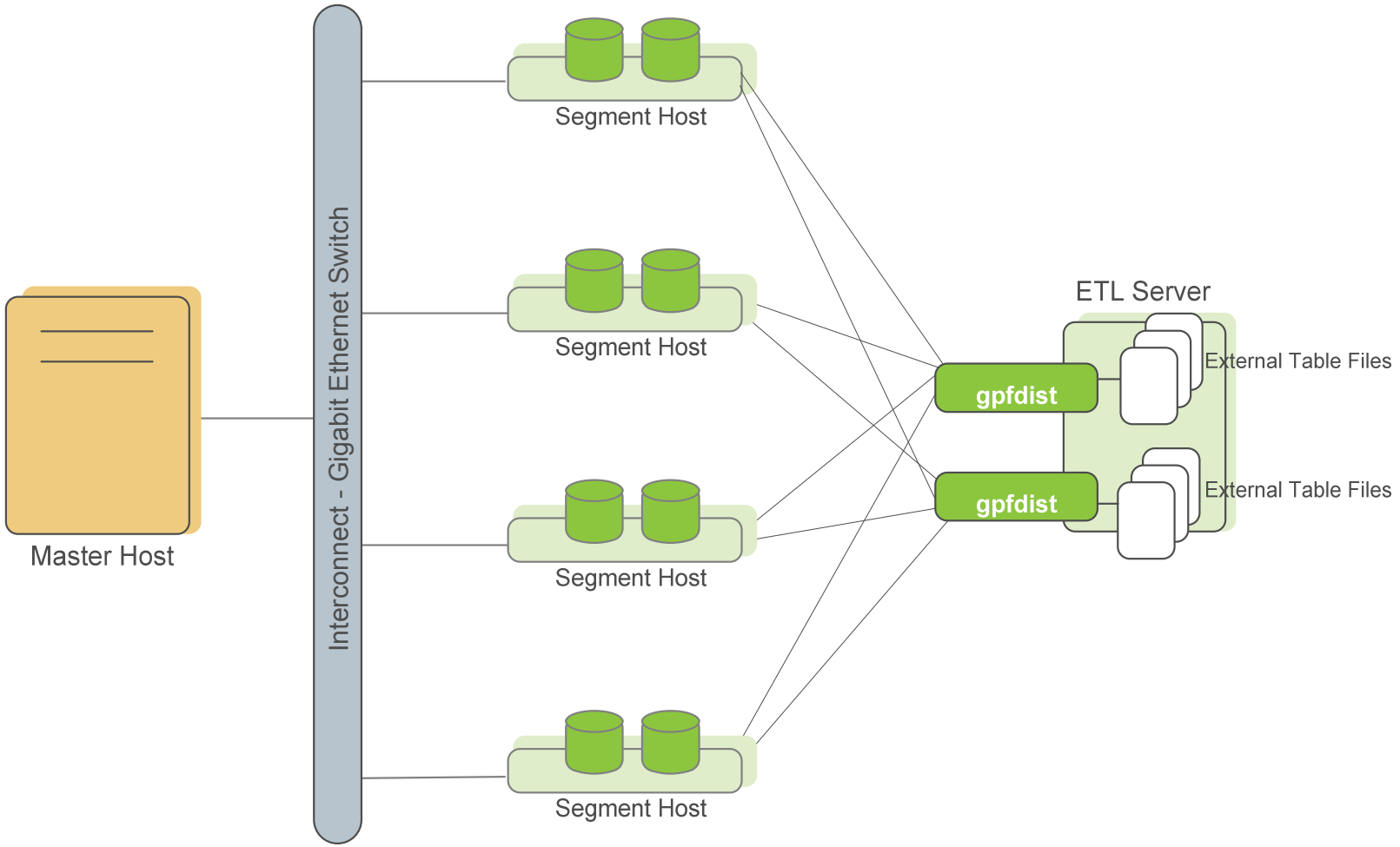
```
gpfdist -d /var/load_files/expenses -p 8081 -l /home/gpadmin/log &
```

- 对应的外部表定义实例:

```
CREATE EXTERNAL TABLE ext_expenses
      ( name text, date date,  amount float4, description text )
LOCATION ('gpfdist//etlhost:8081/*','gpfdist//etlhost:8082/*')
FORMAT 'TEXT' (DELIMITER '|')
ENCODING 'UTF-8'
LOG ERRORS INTO ext_expenses_loaderrors
SEGMENT REJECT LIMIT 10000 ROWS ;
```

Greenplum

并行的文件分布程序(gpfdist)



Greenplum

外部表相关SQL语法

```
CREATE EXTERNAL TABLE table_name (column_name data_type [, ...])

      LOCATION ('file://seghost[:port]/path/file' [, ...])
             | ('gpfdist://filehost[:port]/file_pattern' [, ...])
      FORMAT 'TEXT | CSV'
                [( [DELIMITER [AS] 'delimiter']
                [NULL [AS] 'null string']
                [ESCAPE [AS] 'escape' | 'OFF']
                [HEADER]
                [QUOTE [AS] 'quote']
                [FORCE NOT NULL column [, ...] )]
     [ ENCODING 'encoding' ]
     [ [LOG ERRORS INTO error_table] SEGMENT REJECT LIMIT count [ROWS
   | PERCENT] ]

DROP EXTERNAL TABLE table_name;
```

Greenplum

外部网络表

- 动态数据 — 不能重扫描
- 两种格式：
  - URLs
  - 命令或脚本
    - 执行在规定数量的segment实例上
    - 可设置成不可用：gp_external_enable_exec=off
- 输出数据格式：
  - text
  - csv
- 网络表定义的实例：

```
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 );


CREATE EXTERNAL WEB TABLE log_output (linenum int, message text)
    EXECUTE '/var/load_scripts/get_log_data.sh'
    ON HOST
    FORMAT 'TEXT' (DELIMITER '|');


CREATE EXTERNAL WEB TABLE du_space (storage text)
    EXECUTE 'du -sh'
    ON ALL
    FORMAT 'TEXT';
```

Greenplum

基于命令的网络表定义中的环境变量

- 命令的执行通过Greenplum的超级用户(gpadmin)
- 命令不会使用segment主机的登录环境，在 执行命令中 必须设置好这些环境变量。举例来说：

```
CREATE EXTERNAL WEB TABLE blah (blah text)
EXECUTE 'export BLAH=blah-blah-blah; echo $BLAH'
FORMAT 'TEXT';
SELECT * FROM blah;
    blah
-------------
blah-blah-blah
blah-blah-blah
(2 rows)
```

Greenplum

基于命令的网络表定义中的环境变量

- 附加的Greenplum环境变量
  - $GP_CID - 执行外部网络表任务的会话命令编号
  - $GP_DATABASE - 外部网络表定义所在的数据库名称
  - $GP_DATE - 外部网络表命令执行的日期
  - $GP_MASTER_HOST – 分派系统任务的Greenplum master主机
  - $GP_MASTER_PORT – Greenplum master实例的端口号
  - $GP_SEGMENT_COUNT - 系统中主Segment实例的总数
  - $GP_SEG_DATADIR – 执行命令的segment实例的数据目录
  - $GP_SEG_PG_CONF – Segment实例中postgresql.conf 文件所在位置
  - $GP_SEG_PORT - 执行命令的segment实例 的端口号
  - $GP_SEGMENT_ID – segment的*dbid* (从*pg_catalog.gp_configuration获得*).
  - $GP_SESSION_ID – 与segment相关的数据库会话识别码
  - $GP_SN – 查询计划中节点扫描的序列号
  - $GP_TIME – 外部网络表执行命令的时间
  - $GP_USER – 执行外部网络表任务的数据库用户
  - $GP_XID – 外部网络表任务的事务代码

外部表的错误处理

- 加载正确的数据行，并制止格式不匹配的行，例如：
  - 行的内容不完整或超出定义内容
  - 行中包含与定义数据类型不符合的内容
  - 客户端编码不匹配的行
- 对于限制条件的错误不起作用：
  - PRIMARY KEY, NOT NULL, CHECK 或 UNIQUE 限制条件
- 可选的错误处理语句：

```
[LOG ERRORS INTO error_table] SEGMENT REJECT LIMIT count [ROWS
  | PERCENT]
```

(PERCENT是基于gp_reject_percent_threshold参数)

- 实例：

```
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;
```

Greenplum

外部表和查询计划的统计

- 数据存放在数据库之外
- 对于外部表没有数据库的统计方式
- 意味着不会频繁或实时的读取
- 在系统表pg_class中手动设置粗略的统计：

```
UPDATE pg_class
  SET reltuples=400000, relpages=400
  WHERE relname='myexttable';
```

Greenplum

COPY命令

- PostgreSQL命令
- 加载大数据量的优化方式，相对于INSERT命令
- 通过一个命令加载所有数据行（非并行方式）
- 从一个文件或标准输入加载数据
- 与外部表一样支持错误处理
- 实例：

```
COPY mytable FROM '/data/myfile.csv' WITH
  CSV HEADER;
```

Greenplum

数据加载的性能提示

- 删除索引并在加载后重新创建
  - 增加maintenance_work_mem参数来加快 CREATE INDEX操作的速度
- 加载后运行ANALYZE命令
- 加载出现错误后运行VACUUM 命令
- 不要使用ODBC INSERT来加载大量的数据

第9课练习

数据加载

第10课

数据操作语言(DML)
数据查询语言(DQL)

- Inserts
- Updates
- Deletes
- Selects, OLAP扩展
- 事务处理

Greenplum数据库对SQL的支持

- Greenplum支持SQL-2003标准的绝大部分功能
- Greenplum的目标是完全支持PostgreSQL中所有SQL命令
- DDL命令有部分功能不支持(triggers, foreign keys等)
- 支持所有的DML/DQL 命令 (有某些小的限制条件）
- Greenplum在通用PostgreSQL基础上有针对DW/BI的增强SQL功能 (Parallelism, OLAP, External Tables, Workload Management)

插入数据

- INSERT 命令

  实例：

```
INSERT INTO names VALUES (nextval('names_seq'),
  'test', 'U');
```

- **如果有多行数据需要插入，考虑使用“**INSERT INTO *<table>* SELECT FROM *<external table>*”**或** COPY **命令**

更新数据

- UPDATE 命令 （**Greenplum限制: 任意表的连接必须是equijoins（表具有同样分布键的列而且用这些列进行连接））**

  实例:

  ```
  UPDATE names SET name='Emily' WHERE
  name='Emmmily';
  ```

Greenplum

删除数据

- DELETE命令（**Greenplum限制：任意表的连接必须是equijoins**）

  实例:

  ```
  DELETE FROM rank;
  ```
  (删除所有行)

  ```
  DELETE FROM rank WHERE year='2001';
  ```

- TRUNCATE命令

  实例：

  ```
  TRUNCATE b2001;
  ```
  (删除指定表的所有行)

- DROP命令

  实例：

  ```
  DROP TABLE rank;
  ```
  (删除表的所有行和表结构)

Greenplum

SELECT的限制

- 在correlated subquery结构中会有某些限制
- 在使用VOLATILE 和STABLE类型的函数时有限制

SELECT – OLAP分组扩展

- 标准的GROUP BY语句实例
- ROLLUP
- GROUPING SETS
- CUBE
- grouping(column [, ...])函数
- group_id() 函数

标准的GROUP BY语句实例

根据厂商总结产品销售情况...

```
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)
```

Greenplum

标准的GROUP BY语句实例（续）

包含每个产品小计和总计...

```
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
(19 rows)
```

Greenplum

使用ROLLUP实例

用ROLLUP实现同样的查询...

```
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
(19 rows)
```

Greenplum

GROUPING SETS实例

使用GROUPING SETS实现同样查询...

```
SELECT pn, vn, sum(prc*qty)
FROM sale
GROUP BY GROUPING SETS
      ( (pn, vn), (pn), () )
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
(19 rows)
```

Greenplum

CUBE实例

CUBE会创建表列所有可能组合的小计，所以下面的例子

```
SELECT pn, vn, sum(prc*qty)
FROM sale
GROUP BY CUBE(pn, vn)
ORDER BY 1,2,3;
```

相当于...

```
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 | 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
     | 10 | 0
     | 20 | 0
     | 30 | 180
     | 40 | 2640002
     | 50 | 0
     |    | 2640182
(24 rows)
```

Greenplum

GROUPING函数的实例

从汇总的标签中区分值为空的情况...

```
SELECT * FROM dsales_null;

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);

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)
```

Greenplum

GROUP_ID函数

- 在唯一的grouping set的输出行中返回0
- 如果发现重复的grouping set，依次分配>0的一系列值
- 对于组合的grouping扩展语句有用
- 能够用于过滤重复的grouping sets输出行：

```
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;
```

Greenplum

SELECT – OLAP窗口扩展

- 窗口函数(Window Functions)介绍
- 创建窗口定义
  - OVER语句
  - WINDOW语句
- 内建的窗口函数

窗口函数(Window Functions)介绍

- 只允许在SELECT内容中使用的新类型函数
- 每行返回一个值（不同于聚合函数）
- 结果会按照当前行或行所对应的窗口分隔/框架来说明
- 以OVER语句作为使用的特征
  - 定义窗口分隔（一组行内容）来应用函数
  - 在窗口内定义数据的次序
  - 在涉及的窗口内定义行的位置或逻辑框架

定义窗口的条件 (OVER语句)

- 所有窗口函数都使用OVER()语句
- 在括号内指定应用的函数
- 定义：
  - 窗口分隔 (PARTITION BY语句)
  - 在窗口分隔内的排序 (ORDER BY语句)
  - 在窗口分隔内的图框 (ROWS/RANGE语句)

OVER (PARTITION BY...) 实例

```
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)
```

Greenplum

OVER (ORDER BY...) 实例

```
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)
```

Greenplum

OVER (...ROWS...) 实例

窗口框架："厢式车"方式

```
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)
```

Greenplum

全局窗口条件 (WINDOW语句)

对于多个窗口函数查询有用
定义窗口条件的别名
在整个查询中重复使用这个窗口条件

实例:

```
SELECT
RANK() OVER (ORDER BY pn),
SUM(prc*qty) OVER (ORDER BY pn),
AVG(prc*qty) OVER (ORDER BY pn)
FROM sale;

SELECT
RANK() OVER (w1),
SUM(prc*qty) OVER (w1),
AVG(prc*qty) OVER (w1)
FROM sale
WINDOW w1 AS (ORDER BY pn);
```

内建的窗口函数

- cume_dist()
- dense_rank()
- first_value(*expr*)
- lag(expr [,*offset*] [,*default*])
- last_value(*expr*)
- lead(expr [,*offset*] [,*default*])
- ntile(*expr*)
- percent_rank()
- rank()
- row_number()

- **所有的聚合函数 (使用OVER语句) 也能够作为窗口函数使用**

Greenplum

Greenplum数据库中的事务处理(Transactions)

- 事务处理将多个执行语句打包成一个“全有或全无”的操作
- 事务处理的命令
  - BEGIN 或 START TRANSACTION
  - END 或 COMMIT
  - ROLLBACK
  - SAVEPOINT 或 ROLLBACK TO SAVEPOINT
- psql中的Autocommit模式

  ```
  \set autocommit on|off
  ```
- 两阶段提交的事务处理目前不支持
  - **PREPARE TRANSACTION**
  - **COMMIT PREPARED**
  - **ROLLBACK PREPARED**

Greenplum

事务并发的控制

- **MVCC (多版本并发控制方式)**
- **Greenplum支持SQL标准定义的所有事务隔离级别**
- INSERT/COPY**获取行级别的锁定**
- UPDATE/DELETE**获取表级别的锁定**
- **能使用**LOCK**命令获得特殊的锁定方式**

Greenplum

表级别的锁定模式

- ACCESS SHARE **(SELECT, ANALYZE)**
- ROW SHARE **(SELECT FOR UPDATE, SELECT FOR SHARE)**
- ROW EXCLUSIVE **(INSERT, COPY)**
- SHARE UPDATE EXCLUSIVE **(VACUUM)**
- SHARE **(CREATE INDEX)**
- SHARE ROW EXCLUSIVE
- EXCLUSIVE **(UPDATE/DELETE)**
- ACCESS EXCLUSIVE **(ALTER TABLE, DROP TABLE, REINDEX, CLUSTER, and VACUUM FULL)**

检查锁冲突

- 锁冲突的起因：
  - 并发事务访问同一个对象
  - 资源队列的锁定
  - Segments之间的事务死锁 (罕见)
- 查询pg_locks系统表来查看当前的锁定情况
  实例：

```
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;
```

Greenplum

第10课练习

数据操作语言(DML)
数据查询语言(DQL)

第11课

性能优化

- 性能优化介绍
- 通常的性能问题
- 跟踪性能问题
- 查询程序分析 (EXPLAIN, EXPLAIN ANALYZE)
- 查询优化

性能优化的初始步骤

- 设定性能的期望值
  - 可接受的响应时间，每秒完成的查询数等
  - 对比benchmarks
- 了解基本硬件的性能
  - 读写速度 / 容量
- 了解系统的工作量
  - 使用繁忙的时间段
  - 资源争夺
  - 数据争夺
- 然后关注如何优化

Greenplum

造成性能问题的通常原因

- 硬件问题 / Segments故障
- Resource配置
- 多并发工作量
- 不准确的数据库统计
- 数据分布不均衡
- SQL的编写方式
- 数据库设计

硬件问题

- 磁盘故障
- 主机故障
- 网络故障
- 操作系统没有针对Greenplum调整参数
- 磁盘容量
  - 推荐数据保存不超过磁盘容量的70%
  - 在更新、删除或加载后需要执行
- VACUUM配置参数
  - `max_fsm_relations` = ***tables + indexes + system tables***
  - `max_fsm_pages` = 16 * `max_fsm_relations`

Greenplum

资源的配置和争夺

- 使用Greenplum的资源队列
  - 限制系统的活跃查询数
  - 限制个别用户查询中使用的资源大小
- 将管理工作安排在系统资源相对空闲的时间段
  - 数据加载, ETL
  - `VACUUM, ANALYZE`
  - 备份
- 设计应用程序时避免锁冲突
  - 并发会话不要在同样时间更新同一个数据
- 与资源相关的配置参数
  - `work_mem` = 32MB
  - `maintenance_work_mem` = 64MB
  - `shared_buffers` = 125MB

Greenplum

数据库统计(ANALYZE)

- 数据库的统计用于创建查询计划
- 在下列操作后需运行**ANALYZE:**
  - 数据加载
  - 从备份中恢复数据
  - 修改了模式 (添加了索引等）
  - 插入，更新，删除
- 通过参数配置统计的目标:
  - `default_statistics_target` **= 25**
  - `gp_analyze_relative_error` **= .25**
  - 针对特殊的表列
    - `ALTER TABLE name ALTER column SET STATISTICS #`

Greenplum

Greenplum的数据分布

- 表分布键的考虑因素
  - 均衡的数据分配
  - Segment本地操作还是再分配操作
  - 均衡的查询处理
- 检查数据的分布均衡情况
  - `gpskew -t schema.table`
- 重新设置分布键

```
ALTER TABLE sales set distributed by (Column, …);
```

数据库设计

- 数据类型的选择
  - 尽可能的选择适合数据的最小数据类型
    - 对于整数字段，选择INTEGER而不是 NUMERIC
  - 对于表之间连接列选择同样的数据类型
  - 对于字符型数据，使用varchar 或text
- 非规范化 (星型模式)
- 大表的分区

数据库设计 – 索引

- 在Greenplum数据库中谨慎使用索引
- 首先尝试没有索引的查询
- 索引能提高OLTP类型工作量的性能
- 其它索引方面的考虑：
  - 避免对频繁更新的列创建索引
  - 避免重叠的索引
  - 如果适合，使用位图索引代替B-tree索引
  - 在加载数据前删除索引
  - 考虑聚集索引(clustered index)
- 配置索引使用的参数：
  - `enable_indexscan = on | off`

Greenplum

SQL编写方式

- 综合考虑：
  - 了解数据
  - 返回行数最小化
  - 避免不必要的列/表出现在查询结果中
  - 如果可能，避免大量的分类操作Avoid large sorts if possible
  - 表连接时匹配相应的表类型
- Greenplum的特别考虑：
  - 如果可能，用分布键所在列进行表连接
  - 考虑数据的分布法则和查询语句的对应

跟踪性能问题

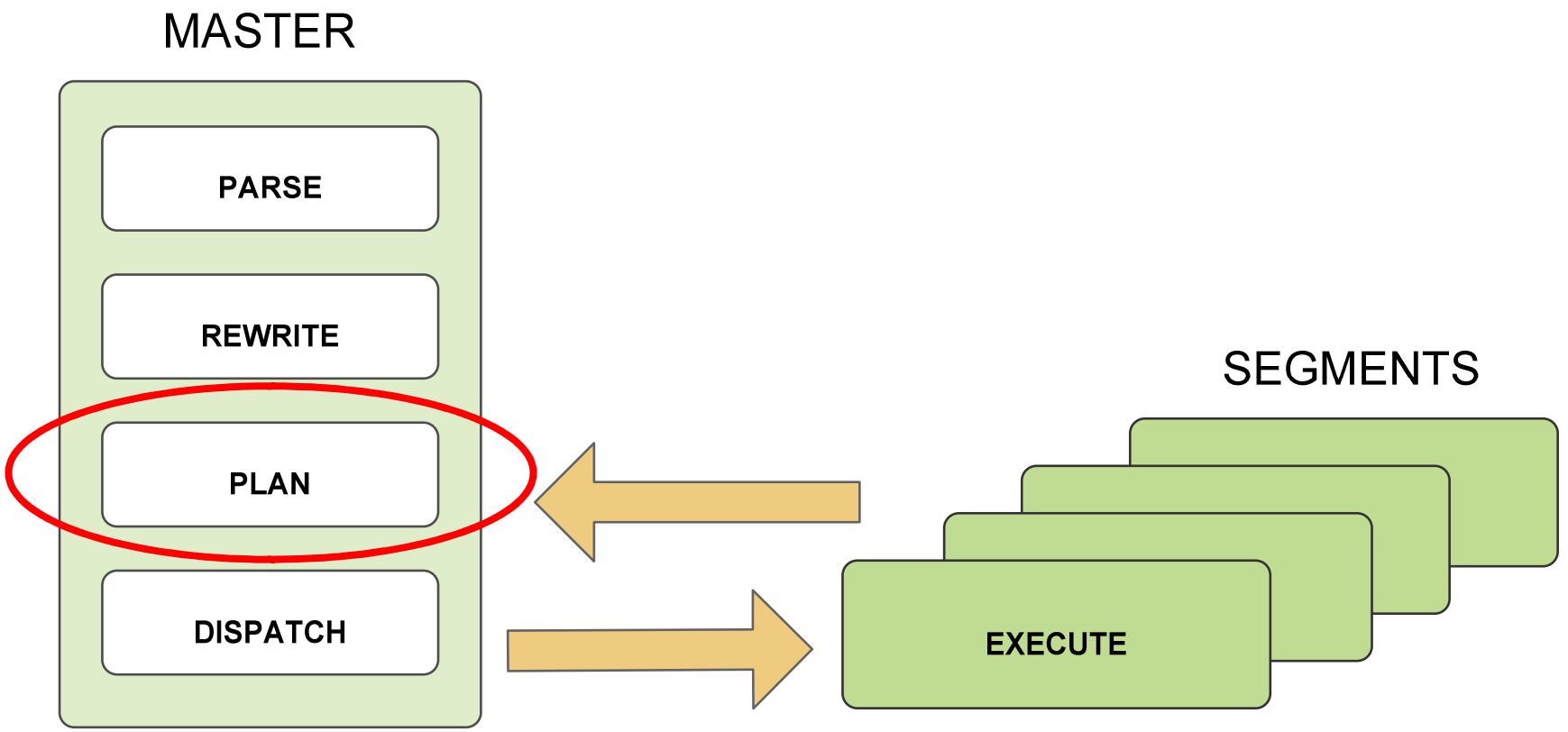
- 相关系统表和视图
  - pg_stat_activity
  - pg_locks / pg_class
- 数据库日志
  - 位于master (和segments)的数据文件夹下
- UNIX系统命令查看 (gpssh)
- Performance Monitor工具

Greenplum

查询的程序分析

- 通过查看查询计划来确定优化的地方
- 可以查看什么?
  - 计划中哪些操作需要执行特别长的时间
  - 查询计划的估算与实际是否接近? (EXPLAIN ANALYZE)
  - 查询计划是否会先应用选择性的语句?
  - 查询计划是否选择最好的连接顺序?
  - 查询计划是否选择性扫描子分区表?
  - 查询计划是否选择了合适的hash聚合和hash连接操作?
  - 是否有足够的work memory?

查询的处理



Greenplum

查看查询计划

- 查看一个查询的计划：
  - `EXPLAIN <query>`
  - `EXPLAIN ANALYZE <query>`
- 查询计划从底往上阅读
  - **Motions (Gather, Redistribute, Broadcast)**
    - **Joins, sorts, aggregations**
      - **Table scans**
- 每个操作都是用以下的度量方式：
  - **cost (读取磁盘页的单位)**
  - **rows (此节点的输出行数)**
  - **width (此节点产生行的字节数)**

Greenplum

阅读EXPLAIN的输出

```
EXPLAIN SELECT * FROM names WHERE name = 'Joelle';

---------------------------- QUERY PLAN ------------------------

Gather Motion 2:1 (slice1) (cost=0.00..20.88 rows=1 width=13)
    -> Seq Scan on 'names' (cost=0.00..20.88 rows=1 width=13)
                Filter: name::text ~~ 'Joelle'::text
```

Greenplum

阅读EXPLAIN ANALYZE的输出

```
EXPLAIN ANALYZE SELECT * FROM names WHERE name = 'Joelle';

---------------------- QUERY PLAN ------------------------

Gather Motion 2:1 (slice1) (cost=0.00..20.88 rows=1 width=13)
  recv: Total 1 rows with 0.305 ms to first row, 0.537 ms to end.
   -> Seq Scan on 'names' (cost=0.00..20.88 rows=1 width=13)
     Total 1 rows (seg0) with 0.255 ms to first row, 0.486 ms to end.
         Filter: name::text ~~ 'Joelle'::text
22.548 ms elapsed
```

Greenplum

查询程序分析和优化的总结

- 数据库统计非常重要(ANALYZE)
- 了解你的数据
  - 列的选择性
  - 行的数量
- 数据库的设计要素是否有助于性能？
  - 数据类型的选择
  - 大表的分区
  - 索引
- 重要的查询优化配置参数
  - work_mem (为分类、连接、聚合设置的工作内存）
  - enable_* (临时取消某个查询操作方式)
  - default_statistics_target, gp_analyze_relative_error (ANALYZE的统计标准)

Greenplum

第11课练习

查询的程序分析

第12课

Greenplum系统管理

- 启动和停止Greenplum数据库
- 检查系统状态
- 检查数据分布情况
- 检查磁盘空间的使用
- 日志文件
- 常规的管理工作清单

启动，停止和重启数据库

- **启动：** `gpstart`
- **停止：** `gpstop`
- **重启：** `gpstop -r`
- **重新加载postgresql.conf:** `gpstop -u`
- **在公用模式下启动/停止master：**

```
gpstart -m
gpstop -m
```

Greenplum

检查系统状态

- 检查系统状态：

  `gpstate`

- 显示完整的系统状况：

  `gpstate -s`

- 查看系统使用的端口：

  `gpstate -p`

- 查看系统镜像的配置：

  `gpstate -m` (只显示镜像的segments)

  `gpstate -c` (主segment与镜像的对应关系)

Greenplum

检查数据的分布情况

- **检查某个表的数据分布情况：**

  `gpskew -t schema.table_name -f`

- **查看某个表的分布键：**

  `gpskew -t schema.table_name -c`

Greenplum

检查磁盘空间的使用

- 检查一个数据库的总容量 (以GB为单位)：

```
gpsizecalc -x db_name -s g
```

- 显示某个表在segments上的大小：

```
gpsizecalc -t schema.table_name
```

- 显示某个表在每个segment上的大小：

```
gpsizecalc -t schema.table_name -f
```

- 检查某个表的索引大小：

```
gpsizecalc -t schema.table_name -i
```

Greenplum

日志文件

- **Greenplum数据库服务器日志文件**
  - **位于数据文件夹下**
  - **在master上**

```
cd $MASTER_DATA_DIRECTORY
```

  - **以及每个segment上**
- **Greenplum管理脚本的日志文件**
  - **位于**`*superuser_home*/gpAdminLogs`

日志的配置参数

- **log_* 服务器配置参数**
  - client_min_messages – 客户端信息
  - log_min_messages – 服务器日志
  - log_connections – 适用于新的连接点
  - log_statement
  - log_duration – 显示数据库日志内的所有时间段，有效地进行查询程序分析

Greenplum

Greenplum服务器日志的故障检修

- **先检查master的日志，发现相关的日志记录**
- **日志行有如下的格式：**

```
timestamp | user | database | statement_id | con#
cmd# |:-MESSAGE _TYPE: <log_message>
```

例如：

```
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;
```

- **基于*statement_id*信息或*con#*和*cmd#*来搜索segment上的相关日志记录**
- **例如：**

```
gpstate -e -r 10 -g “11085”
```

Greenplum

常规的管理工作

- VACUUM 和ANALYZE
  - 不推荐使用VACUUM FULL
  - 在系统相对空闲的时间段执行
  - Greenplum数据库不推荐使用*autovacuum*后台程序
- REINDEX
- 检查Greenplum服务器日志文件

第12课练习

Greenplum管理工作

第13课

数据冗余和高可用性

- 故障运行模式
- 镜像segments
- 故障检测和恢复
- 系统重建
- Segment节点扩容
- 热备master

故障运行模式

- 故障运行模式– *read-only* 或*continue*
  - *Read-only*意味着当一台segment故障时，DML和DDL是不允许的
  - *Continue*意味着当一台segment故障时，所有的SQL操作都可以执行
  - gp_fault_action参数
- Segment状态 – valid 或 invalid
- Segment 恢复方式取决于故障运行模式
  - Read-only模式下不需要关闭系统就能恢复
  - Continue模式下需要关闭系统才能恢复

镜像Segments

- 镜像的segment：
  - 已知主segment的一个复制
  - 实现数据冗余
- 镜像创建的方式：
  - 在系统初始化时创建 (设置`gp_init_config`中的参数)
  - 在已有的Greenplum数据库中添加：`gpaddmirrors`
- 镜像segments可以被配置在：
  - 与主segment位于同一台主机上
  - 与主segment位于不同的主机上

故障检测和恢复

- 故障检测在Greenplum数据库中并不是自动的
  - Segment出问题时不会立刻被标识无效，而是在一个连接尝试失败后才被标识
  - 当一个连接第一次检测到segment有问题时，该连接会失败
  - 如果镜像被激活，随后的连接请求会自动切换到镜像segment上并成功连接
- Greenplum数据库管理员 必须注意及时将标记为*无效* 的segment进行恢复 (`gprecoverseg`)
- Greenplum数据库管理员应当积极地定时检查系统的状态

Greenplum

系统的重建

- 当需要重建数据库系统时：
  - 使用不同的硬件资源
  - 损失了整个系统，需要从备份文件中重新创建
  - 新的系统应当与原系统具有同样数量的segments
  - 重建命令：`gprebuildsystem`

Segment节点扩容

- 当需要增加segment节点实现系统扩容：
  - Greenplum3.3版本支持不用停止系统的扩容功能
  - 使用命令：gpexpand
- 系统扩容的步骤
  - 添加并检查新的硬件平台
  - 初始化新的segment：gpexpand –f new_hosts_file
  - 重新分布数据表（利用系统空闲时间）

Greenplum

热备Master

- 热备的master：
  - 是Greenplum Master实例的备份（系统表）
  - 实现无单点故障
- 热备master的创建方式：
  - 在系统初始化时 (-s选项)
  - 在已有的系统下添加：`gpinitstandby`
- 如何激活热备master：
  - 在活跃的主机上运行`gpactivatestandby`命令

Greenplum

第13课练习

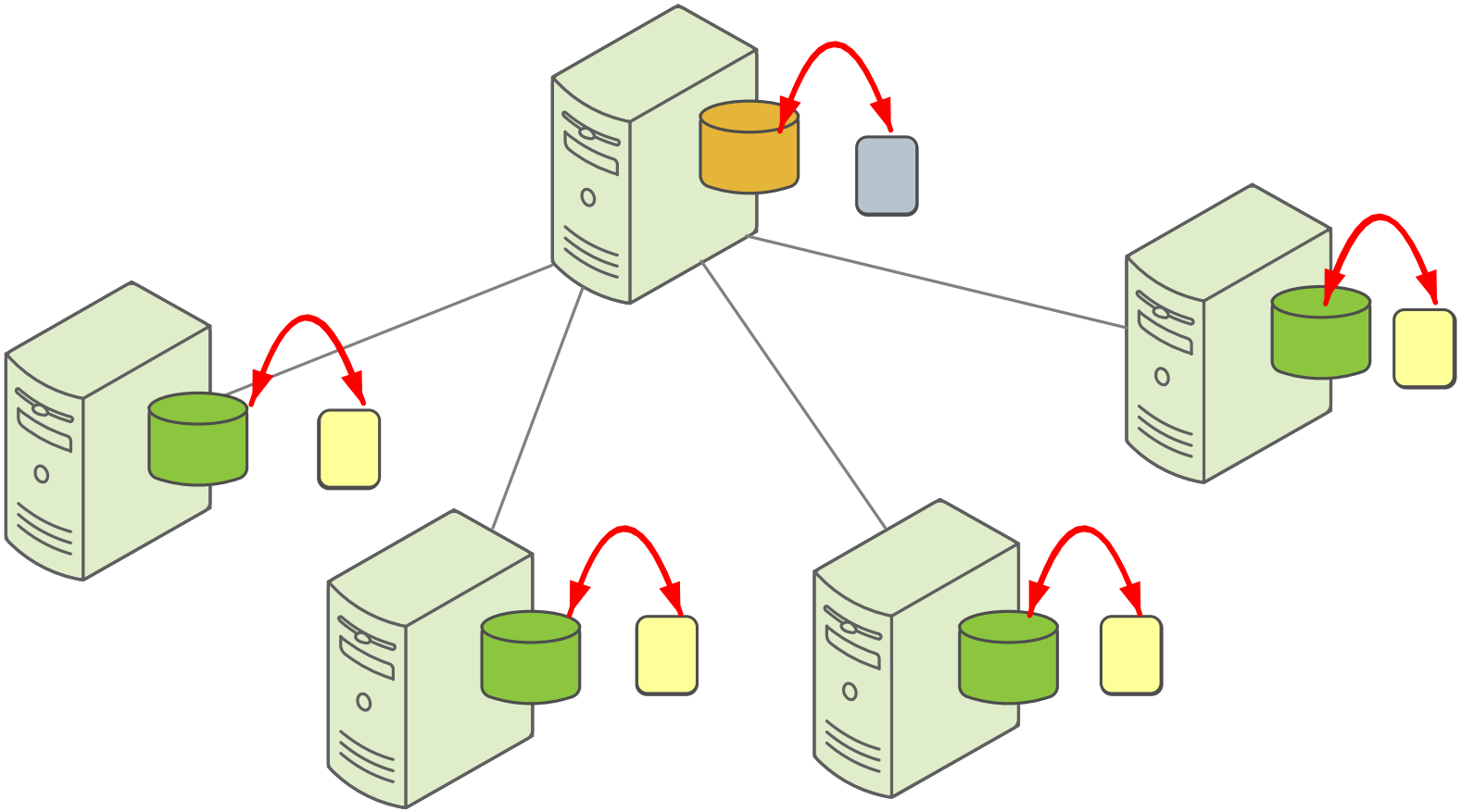
高可用性功能

第14课

备份和恢复

- 并行备份 (gp_dump)
- 并行恢复 (gp_restore)
- 自动备份/恢复 (gpcrondump, gpdbrestore)
- 非并行的备份 (pg_dump)

并行备份和恢复图例



Greenplum

运行并行备份命令(gp_dump)

- 用并行的方式在每个segment上创建备份
- 备份文件默认创建在数据文件夹下
- 支持压缩方式 (gzip)
- 备份前确保服务器上有足够的磁盘空间
- 一个备份集使用唯一的时间戳代码来识别

gp_dump创建的备份文件

在master主机上

- gp_catalog_1_<dbid>_<timestamp>
- gp_cdatabase_1_<dbid>_<timestamp>
- gp_dump_1_<dbid>_<timestamp>
- gp_dump_status_1_<dbid>_<timestamp>

在segment主机上

- gp_dump_0_<dbid>_<timestamp>
- gp_dump_status_0_<dbid>_<timestamp>

Greenplum

运行并行恢复命令 (gp_restore)

- 数据恢复使用gp_restore命令
- 需要使用gp_dump创建的时间戳代码
- 恢复前确认备份文件位于每台segment主机的正确位置
- 恢复前确认数据库已存在
- 数据库级别的服务器配置不会被恢复

定时的常规备份

- gpcrondump
  - 调用gp_dump
  - 使用CRON直接调用或者根据计划表
  - 发送电子邮件通知
  - 灵活的备份选项
  - 可复制配置文件
  - 可备份系统目录
  - 可备份全局对象
  - 可包含备份脚本
- Gpdbrestore
  - 从gpcrondump生成的文件恢复
  - 可从归档的服务器上恢复—不需要预先将备份放到segments上

Greenplum

非并行的备份和恢复

- Greenplum也支持pg_dump和pg_restore
- 对于移植数据从其他数据库或到其他数据库有用
- pg_dump创建一个单一的备份文件
  - 对于非常大的数据库速度较慢
  - 运行在低使用率的时间
  - 支持压缩功能
  - 应用INSERT或COPY命令备份数据
  - --gp-syntax选项中包括在DDL中的DISTRIBUTED BY 语句

第14课练习

备份和恢复

第15课

数据库内部结构

- 系统目录表
- 物理存储
- 服务器进程

系统目录表和视图

- 所有的系统目录都在pg_catalog schema下
- 标准的PostgreSQL系统目录 (pg_*)
- Greenplum特色的目录对象:
  - gp_configuration
  - gp_distribution_policy
  - gp_id
  - gp_version_at_initdb
  - pg_resqueue
  - pg_exttable
- 在psql中显示系统表清单： \dtS
- 在psql中显示系统视图清单： \dvS

Greenplum

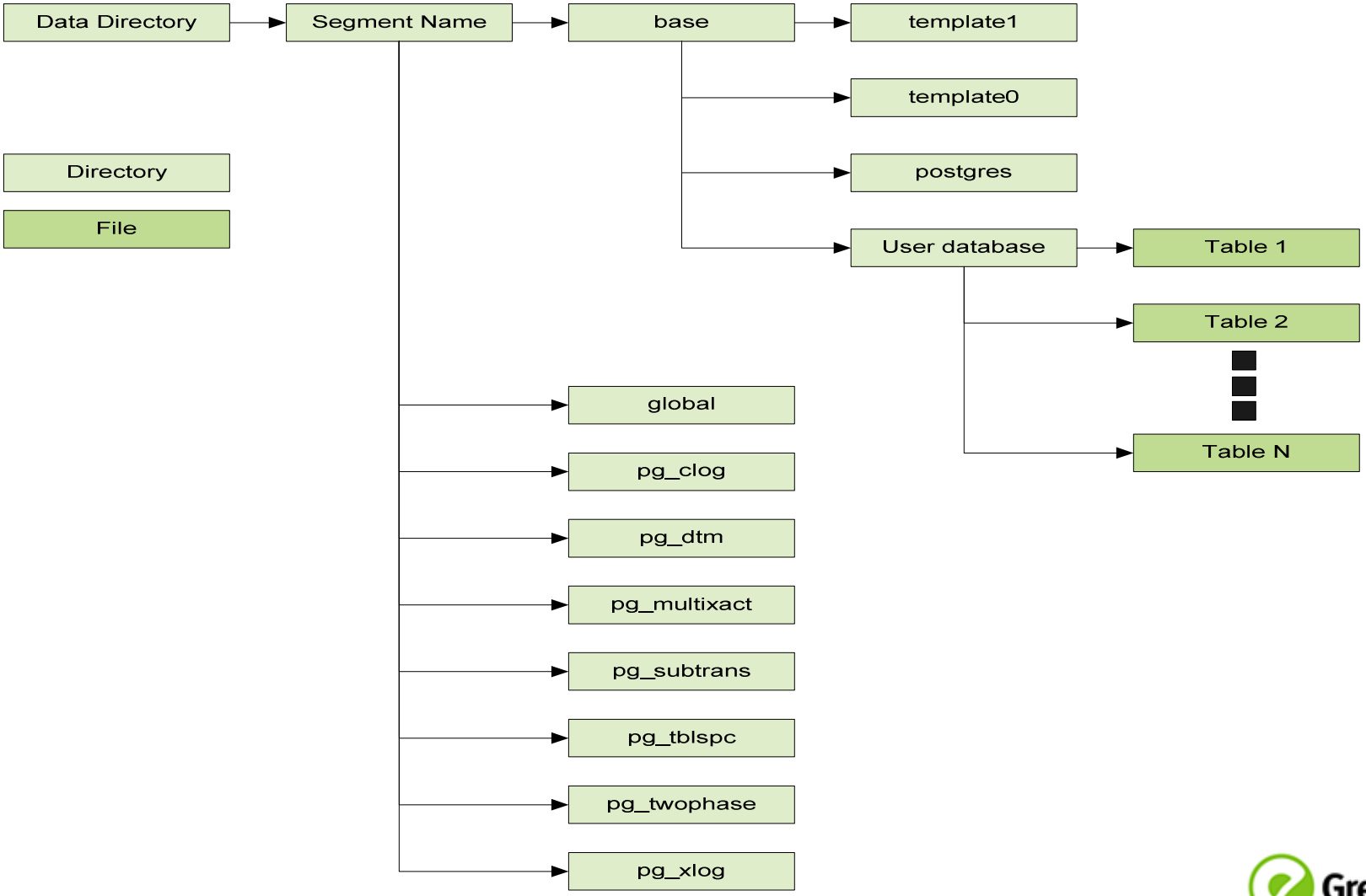
统计信息收集

- 收集数据库活动的相关信息
- 服务器配置参数
  - start_stats_collector = on
  - stats_block_level = off
  - stats_row_level = off
  - stats_queue_level = off
  - stats_command_string = on
- 查看系统目录中的统计视图和表：

```
\dtvS pg_stat*
```

Greenplum

物理存储— 数据文件夹的文件结构



Greenplum

服务器进程

- **Linux:** `ps ax | grep postgres`
- **Solaris:** `ps -ef | grep postgres`
  `pargs <process_id>`

Greenplum Master实例

- **Postgres数据库的监听进程**

```
postgres: <sub_process_name>
postgres: seqserver process
postgres: <user><database><con#><host><cmd#><slice#>
```

Greenplum Segment实例

- **postgres数据库的监听进程**

```
postgres: <sub_process_name>
```

Greenplum

第15课练习

数据库内部结构