

Greenplum 数据库调优

目录

Greenplum 数据库调优.....	1
目录.....	1
1 Greenplum 查询处理回顾.....	2
1.1 Master 把查询语句分发到 segment.....	2
2 Greenplum 数据库调优.....	3
2.1 系统资源.....	3
2.2 硬件问题.....	4
2.3 资源管理.....	5
2.3.1 查看 resource queue 的参数.....	5
2.3.2 设置临时的内存大小.....	6
2.3.3 当发生数据溢出时添加内存的大小.....	6
2.3.4 受影响的系统的参数.....	7
2.3.5 查看一些有用的视图.....	7
2.4 统计信息不准确.....	8
2.4.1 创建两张表.....	8
2.4.2 使用 EXPLAIN 查看执行计划.....	8
2.4.3 使用 ANALYZE 执行统计信息.....	9
2.4.4 以下情况都需要执行 ANALYZE.....	10
2.5 数据倾斜.....	10
2.5.1 数据倾斜实例.....	10
2.5.2 使用视图查看表的倾斜.....	11
2.5.3 改变数据倾斜问题.....	11
2.6 计算倾斜.....	12
2.6.1 关联条件倾斜.....	12
2.6.2 多计算聚集.....	12
2.6.3 减少计算倾斜问题.....	13
2.7 数据广播.....	13
2.7.1 查看表是不是出现了 Broadcast.....	13
2.7.2 改变 planner 之后运行.....	14
2.7.3 修改 GUC 来设定优化器.....	14
2.8 多阶段聚集.....	15
2.8.1 多阶段聚集关闭的情况.....	15
2.8.2 多阶段聚集打开的情况.....	15
2.8.3 GUC 会影响优化器对多阶段聚集的选择.....	16
2.9 分区裁剪.....	17
2.9.1 定义分区.....	17

2.9.2 使用查询计划查看分区.....	17
2.9.2.1 planner 分区.....	17
2.9.2.2 ORCA 分区.....	18
2.9.2.3 最优查询条件.....	18
3 一些最佳实战.....	19
3.1 最佳实践注意点.....	19

1 Greenplum 查询处理回顾

1.1 Master 把查询语句分发到 segment

- 1、Master 接受查询语句并生成查询计划
- 2、Master 把查询计划分发到 Segment，分发模式有两种分别是 Parallel 和 Targeted
- 3、Segment 并发在各自本地的数据集上执行计划
 - a、Slice : A portion of the plan that segment can work on independently
 - b、Gang : Related processes that are working on the same slice of the query plan but on different segments
- 4、Master 收集结果并返回给客户端

Greenplum查询处理回顾

Greenplum 对查询的处理过程可以分成四个步骤:

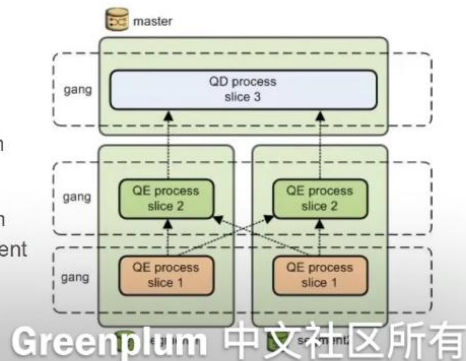
1. Master 接收查询语句并生成查询计划。
2. Master 把查询计划分发到Segments.



Greenplum查询处理回顾

Greenplum 对查询的处理过程可以分成四个步骤:

1. Master 接收查询语句并生成查询计划。
2. Master 把查询计划分发到Segments.
3. Segments 并发在各自本地的数据集上 执行计划。
 - a. Slice: A portion of the plan that segments can work on independently.
 - b. Gang: Related processes that are working on the same slice of the query plan but on different segments.
4. Master 收集结果并返回给客户端。



2 Greenplum 数据库调优

使用 EXPLAIN ANALYZE 查看执行计划

```
# EXPLAIN ANALYZE select * from t1 join t2 on t1.c1 = t2.c2;
```

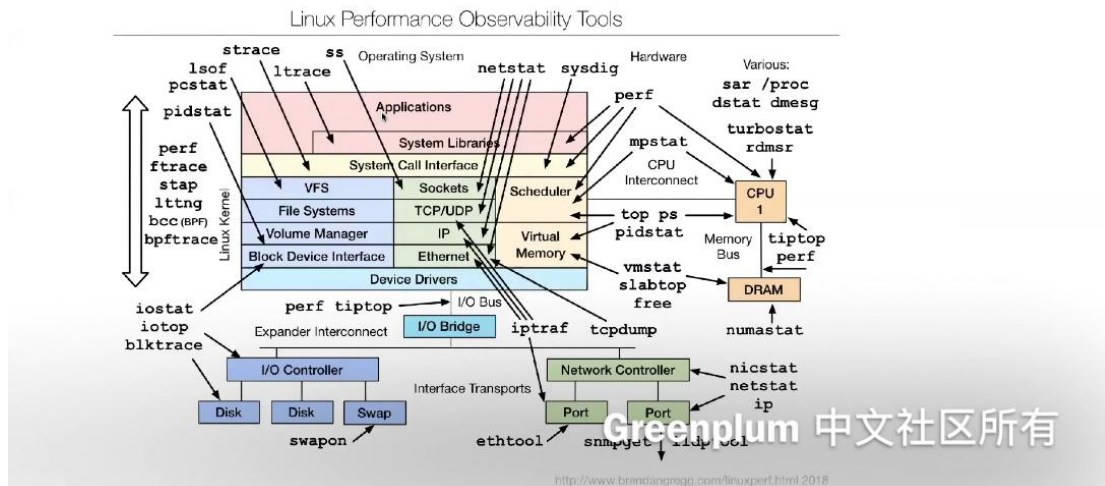
Gather Motion 3:1 (slice2; segments: 3) (cost=0.00..1197.67 rows=1000809 width=16)
Rows out: 1000000 rows at destination with 519 ms to first row, 1596 ms to end.
-> **Hash Join** (cost=0.00..1138.00 rows=333603 width=16)
Hash Cond: t1.c1 = t2.c2
Rows out: Avg 333333.3 rows x 3 workers. Max 333385 rows (seg2) with 517 ms to first row, 1254 ms to end.
Executor memory: 10417K bytes avg, 10419K bytes max (seg2).
Work_mem used: 10417K bytes avg, 10419K bytes max (seg2). Workfile: (0 spilling)
(seg2) Hash chain length 1.4 avg, 7 max, using 246546 of 524288 buckets.
-> **Table Scan on t1** (cost=0.00..444.94 rows=667206 width=8)
Rows out: Avg 666666.7 rows x 3 workers. Max 666720 rows (seg2) with 0.094 ms to first row, 111 ms to end.
-> **Hash** (cost=451.29..451.29 rows=333603 width=8)
Rows in: Avg 333333.3 rows x 3 workers. Max 333385 rows (seg2) with 492 ms to end, start offset by 58 ms.
-> **Redistribute Motion 3:3** (slice1; segments: 3) (cost=0.00..451.29 rows=333603 width=8)
Hash Key: t2.c2
Rows out: Avg 333333.3 rows x 3 workers at destination. Max 333385 rows (seg2) with 0.280 ms to first row, 299 ms to end
-> **Table Scan on t2** (cost=0.00..437.97 rows=333603 width=8)
Rows out: Avg 333333.3 rows x 3 workers. Max 333385 rows (seg2) with 0.148 ms to first row, 47 ms to end.
Slice statistics:
(slice0) Executor memory: 322K bytes.
(slice1) Executor memory: 191K bytes avg x 3 workers, 191K bytes max (seg0).
(slice2) Executor memory: 24875K bytes avg x 3 workers, 24875K bytes max (seg0). Work_mem: 10419K bytes max.
Statement statistics:
Memory used: 128000K bytes
Optimizer status: PQO version 3.63.1
Total runtime: 1713.756 ms
(25 rows)

Greenplum 中文社区所有

2.1 系统资源

按照不同的问题使用不同的 shell 命令

系统资源



2.2 硬件问题

使用 Greenplum 自带的 `gpcheckperf` 命令检测硬件问题

硬件问题

常见的硬件问题包括：

- 磁盘 I/O 问题
- 网络性能问题
- 内存带宽问题

硬件问题

```
gpcheckperf -d test_directory [-d test_directory ...]  
{-f hostfile_gpcheckperf | -h hostname [-h hostname ...]}  
[-r ds] [-B block_size] [-S file_size] [-D] [-v|-V]  
  
gpcheckperf -d temp_directory  
{-f hostfile_gpchecknet | -h hostname [-h hostname ...]}  
[-r n|N|M [--duration time] [--netperf] ] [-D] [-v|-V]  
  
gpcheckperf -?  
  
gpcheckperf --version
```

Greenplum 中文社区所有

2.3 资源管理

2.3.1 查看 resource queue 的参数

资源管理主要与 resource queue 中的 PRIORITY 和 MEMORY_LIMIT 有关

资源管理: resource queue

- PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX}
 - 此资源队列中查询的优先级。
 - 在有争用的情况下, 具有较高优先级的队列中的查询将获得更大的可用 CPU 资源份额。
- MEMORY_LIMIT
 - 此资源队列中所有查询的总内存配额。

Greenplum 中文社区所有

2.3.2 设置临时的内存大小



资源管理: resource queue

每个查询所能使用的内存量可以被GUC 'statement_mem' 覆盖。所以, 对于某一个查询, 如果想使用更多的内存, 可以这样做:

```
=> SET statement_mem='2GB';

=> SELECT * FROM my_big_table WHERE column='value'
ORDER BY id;

=> RESET statement_mem;
```

Greenplum 中文社区所有

2.3.3 当发生数据溢出时添加内存的大小

使用 EXPLAIN ANALYZE 查看执行计划时发现 Work_mem used 比 Work_mem wanted 小时, 可能数据文件溢出到磁盘上, 需要临时添加内存



资源管理: resource queue

当执行节点发生数据溢出时, 可以尝试增加 'statement_mem'.

```
-> Hash Join (cost=23619.20..115455.80 rows=655521 width=16)
    Hash Cond: t2.c2 = t1.c1
    Rows out: Avg 666664.7 rows x 3 workers. Max 666770 rows (seg2) with 243 ms
to first row, 2668 ms to end.
    Executor memory: 101K bytes avg, 101K bytes max (seg0).
    Work_mem used: 101K bytes avg, 101K bytes max (seg0). Workfile: (3 spilling)
    Work mem wanted: 10417K bytes avg, 10419K bytes max (seg2) to lessen workfile
I/O affecting 3 workers.
    (seg2) Initial batch 0:
    (seg2) Wrote 6096K bytes to inner workfile.
    (seg2) Wrote 12192K bytes to outer workfile.
```

Greenplum 中文社区所有

2.3.4 受影响的系统的参数



资源管理: resource group

- CPU_RATE_LIMIT
 - 分配给此资源组的CPU资源的百分比。
- MEMORY_LIMIT
 - 为此资源组预留的内存资源的百分比。

Greenplum 中文社区所有

2.3.5 查看一些有用的视图

首先系统视图 pg_stat_activity 和 pg_locks 来查看当前是否有锁的等待



资源管理: 一些有用的视图

pg_stat_activity

```
-[ RECORD 1 ]-----+-----
datid        | 16384
datname      | gpadmin
procpid      | 31724
sess_id     | 18
usesysid    | 10
username    | gpadmin
current_query | select * from t1 join t2 on t1.c1 = t2.c1;
waiting      | t
query_start  | 2019-08-13 05:13:03.023949-04
backend_start | 2019-08-13 04:14:26.949737-04
client_addr  |
client_port  | -1
application_name | psql
xact_start   | 2019-08-13 05:13:03.023949-04
waiting_reason | lock
rsgid       | 0
rsgname     | unknown
rsgqueueduration |
```

Greenplum 中文社区所有

资源管理:一些有用的视图

pg_locks

locktype	database	relation	pid	mode	granted	mppsessionid	gp_segment_id
relation	16384	1247	31831	RowExclusiveLock	t	20	-1
relation	16384	1259	31831	RowExclusiveLock	t	20	-1
relation	16384	2608	31831	RowExclusiveLock	t	20	-1
relation	16384	5002	31831	RowExclusiveLock	t	20	-1
relation	16384	5010	31831	RowExclusiveLock	t	20	-1
relation	16384	5011	31831	RowExclusiveLock	t	20	-1
relation	16384	11343	31846	AccessShareLock	t	21	-1
relation	16384	16769	31831	AccessExclusiveLock	t	20	-1
relation	16384	16769	31724	AccessShareLock	f	18	-1

Greenplum 中文社区所有

2.4 统计信息不准确

2.4.1 创建两张表

统计信息不准确

Greenplum依赖统计信息做出最优的plan。当统计信息不准确时,会影响性能。

```
# select count(*) from t1;
count
-----
 1000000
(1 row)

# select count(*) from t2;
count
-----
 100000
(1 row)
```

Greenplum 中文社区所有

2.4.2 使用 EXPLAIN 查看执行计划

在以下的计划中可以看出全表扫描了并 hash 了表 t1,并返回 rows=1 行数据,耗时 851.390ms

统计信息不准确

```
# EXPLAIN select * from t1 join t2 on t1.c1 = t2.c1;
```

QUERY PLAN

```
-----  
Gather Motion 3:1 (slice1; segments: 3) (cost=1.01..1356.81 rows=17 width=16)  
-> Hash Join (cost=1.01..1356.81 rows=6 width=16)  
    Hash Cond: t2.c1 = t1.c1  
    -> Seq Scan on t2 (cost=0.00..1106.70 rows=33190 width=8)  
    -> Hash (cost=1.00..1.00 rows=1 width=8)  
        -> Seq Scan on t1 (cost=0.00..1.00 rows=1 width=8)  
Optimizer status: legacy query optimizer  
(7 rows)
```

Total runtime: 851.390 ms

Greenplum 中文社区所有

2.4.3 使用 ANALYZE 执行统计信息

在以下中可以看出执行了 analyze 后，全表扫描和 Hash 了 t2,返回了 rows=33325 行数据,耗时 553.725ms

统计信息不准确

```
# ANALYZE ;  
ANALYZE
```

```
# EXPLAIN select * from t1 join t2 on t1.c1 = t2.c1;
```

QUERY PLAN

```
-----  
Gather Motion 3:1 (slice1; segments: 3) (cost=2360.44..17221.24 rows=99975 width=16)  
-> Hash Join (cost=2360.44..17221.24 rows=33325 width=16)  
    Hash Cond: t1.c1 = t2.c1  
    -> Seq Scan on t1 (cost=0.00..11109.09 rows=333603 width=8)  
    -> Hash (cost=1110.75..1110.75 rows=33325 width=8)  
        -> Seq Scan on t2 (cost=0.00..1110.75 rows=33325 width=8)  
Optimizer status: legacy query optimizer  
(7 rows)
```

Total runtime: 553.725 ms

Greenplum 中文社区所有

2.4.4 以下情况都需要执行 ANALYZE



统计信息不准确

一般来说, 当有下面几种行为发生时, 需要用ANALYZE来更新统计信息:

- 数据加载 (COPY, Gpload, gpfdisk, ...)
- 大量 INSERT, UPDATE, 和 DELETE 操作
- CREATE INDEX
- 修改schema
- 数据库从备份中恢复
- ...

Greenplum 中文社区所有

2.5 数据倾斜

2.5.1 数据倾斜实例

使用 EXPLAIN ANALYZE 出现以下信息说明执行了 3 个 workers, 每个 workers 平均执行了 366666.7 行的数据, 在 seg0 上最大执行了 1033348 行数据, 说明大部分的数据都在 seg0 上, 有数据倾斜的现象。



数据倾斜

当数据在各个Segments之间分布不平衡时, 就会发生数据倾斜, 影响系统整体性能。

```
# EXPLAIN ANALYZE select * from t1 join t2 on t1.c1 = t2.c1;
```

.....

```
-> Seq Scan on t1 (cost=0.00..12209.91 rows=366631 width=8)
    Rows out: Avg 366666.7 rows x 3 workers. Max 1033348 rows (seg0) with 0.045
ms to first row, 126 ms to end.
```

Greenplum 中文社区所有

2.5.2 使用视图查看表的倾斜

使用 `gp_skew_coefficients` 和 `gp_skew_idle_fractions` 来查看表的倾斜的情况，其中 `gp_skew_coefficients` 是标准偏差除以平均值，值越低越好。`gp_skew_idle_fractions` 是表扫描期间空闲的系统百分比，超过 10% 的表应该评估表的分布策略。



数据倾斜

相关视图

```
# select * from gp_toolkit.gp_skew_coefficients;
skcoid | skcnamespace | skcrelname |      skccoeff
-----+-----+-----+-----
  16401 | public       | t1         | 157.46262844967713000
  16404 | public       | t2         | 0.048497422611928613000
(2 rows)
```

标准偏差除以平均值。值越低越好。

```
# select * from gp_toolkit.gp_skew_idle_fractions;
sifoid | sifnamespace | sifrelname |      siffraction
-----+-----+-----+-----
  16401 | public       | t1         | 0.64516632667149240462
  16404 | public       | t2         | 0.00043980648514653552
(2 rows)
```

表扫描期间空闲的系统百分比。超过10%的表应评估其分配策略。

Greenplum 中文社区所有

2.5.3 改变数据倾斜问题

使用改变分布键来改变数据倾斜的问题



数据倾斜

重新选择分布键

```
ALTER TABLE name SET DISTRIBUTED BY (column ...)
```

Greenplum 中文社区所有

2.6 计算倾斜

2.6.1 关联条件倾斜

当进行关联条件关联时，两个关联建如果分配不均匀的话可能也会出现倾斜



计算倾斜

当数据经过重分布后可能会出现各segments之间分布不平衡，此时就会发生计算倾斜。

```
# EXPLAIN ANALYZE select * from t1 join t2 on t1.c2 = t2.c2;
```

QUERY PLAN

```
-----
Gather Motion 3:1 (slice3; segments: 3) (cost=4381.32..50291.52 rows=995760 width=16)
  Rows out: 1100000 rows at destination with 63 ms to first row, 2612 ms to end.
  -> Hash Join (cost=4381.32..50291.52 rows=331920 width=16)
    Hash Cond: t2.c2 = t1.c2
    Rows out: Avg 550000.0 rows x 2 workers. Max 1000000 rows (seg0) with 61 ms to first row, 2219
    ms to end.
```

Greenplum 中文社区所有

2.6.2 多计算聚集

多计算倾斜就是经过了第一步的处理又经过了第二部的处理，如下例:经过了 group by 之后有进行了 avg 计算



计算倾斜

```
# EXPLAIN ANALYZE select avg(distinct c2) from t2 group by c2;
```

QUERY PLAN

```
-----
Gather Motion 3:1 (slice2; segments: 3) (cost=64181.68..64591.52 rows=32787 width=36)
  Rows out: 33448 rows at destination with 977 ms to first row, 1070 ms to end.
  -> HashAggregate (cost=64181.68..64591.52 rows=10929 width=36)
    Group By: t2.c2
    Rows out: Avg 16724.0 rows x 2 workers. Max 33348 rows (seg0) with 1020 ms to first row, 1059
    ms to end.
```

Greenplum 中文社区所有

2.6.3 减少计算倾斜问题



计算倾斜

需要综合考虑以下方面, 尽量减少计算过程中产生的倾斜问题:

- 分布键
- 连接条件
- 分组键

Greenplum 中文社区所有

2.7 数据广播

2.7.1 查看表是不是出现了 Broadcast




数据广播

```
# EXPLAIN select * from t1 join t2 on t1.c2 = t2.c1;
                                QUERY PLAN
-----
Gather Motion 3:1 (slice2; segments: 3) (cost=43635.38..83367.72 rows=1000809 width=16)
-> Hash Join (cost=43635.38..83367.72 rows=333603 width=16)
    Hash Cond: t1.c2 = t2.c1
    -> Seq Scan on t1 (cost=0.00..22218.18 rows=667206 width=8)
    -> Hash (cost=31125.27..31125.27 rows=333603 width=8)
        -> Broadcast Motion 3:3 (slice1; segments: 3) (cost=0.00..31125.27
rows=333603 width=8)
            -> Seq Scan on t2 (cost=0.00..11109.09 rows=333603 width=8)
        Settings: gp segments for planner=1
        Optimizer status: legacy query optimizer
        (9 rows)
```

Total runtime: 3391.539 ms

Greenplum 中文社区所有

2.7.2 改变 planner 之后运行



数据广播

```
# EXPLAIN select * from t1 join t2 on t1.c2 = t2.c1;
                                QUERY PLAN
-----
Gather Motion 3:1  (slice2; segments: 3)  (cost=23619.20..103383.90 rows=1000809 width=16)
-> Hash Join  (cost=23619.20..103383.90 rows=333603 width=16)
    Hash Cond: t1.c2 = t2.c1
    -> Redistribute Motion 3:3  (slice1; segments: 3)  (cost=0.00..62250.54
        rows=667206 width=8)
        Hash Key: t1.c2
        -> Seq Scan on t1  (cost=0.00..22218.18 rows=667206 width=8)
    -> Hash  (cost=11109.09..11109.09 rows=333603 width=8)
        -> Seq Scan on t2  (cost=0.00..11109.09 rows=333603 width=8)

Settings: gp_segments_for_planner=100
Optimizer status: legacy query optimizer
(10 rows)
```

Total runtime: 2116.574 ms

Greenplum 中文社区所有

2.7.3 修改 GUC 来设定优化器

如果配置了 `gp_segment_for_planner` 后，优化器会选择最优的一个来执行



数据广播

在GPDB中，可以通过修改以下GUC来设定优化器在计算代价时假定的segments数量。

- `optimizer_segments`: for ORCA
- `gp_segments_for_planner`: for legacy planner

Greenplum 中文社区所有

2.8 多阶段聚集

2.8.1 多阶段聚集关闭的情况

以下是把 `gp_enable_multiphase_agg` 参数关闭的情况下，耗时 13770.833ms,数据库默认的这个参数是打开的。

多阶段聚集

Greenplum支持多阶段聚集，可以增加聚集操作的并行度，从而提升性能。

```
# select count(*) from t1;
count
-----
2000000
(1 row)
```

Greenplum 中文社区所有

多阶段聚集

```
# EXPLAIN select avg(distinct c2) from t1 group by c2;
QUERY PLAN
-----
Gather Motion 3:1 (slice2; segments: 3) (cost=271747.24..311779.60 rows=2001618 width=36)
-> GroupAggregate (cost=271747.24..311779.60 rows=667206 width=36)
    Group By: c2
    -> Sort (cost=271747.24..276751.28 rows=667206 width=4)
        Sort Key: c2
        -> Redistribute Motion 3:3 (slice1; segments: 3) (cost=0.00..62250.54 rows=667206 width=4)
            Hash Key: c2
            -> Seq Scan on t1 (cost=0.00..22218.18 rows=667206 width=4)
                Settings: gp_enable_multiphase_agg=off
                Optimizer status: legacy query optimizer
                (10 rows)
```

Total runtime: 13770.833 ms

Greenplum 中文社区所有

2.8.2 多阶段聚集打开的情况

以下是 `gp_enable_multiphase_agg` 打开的情况下，运行了三次聚集，运行耗时 6041.057ms

多阶段聚集

```
# EXPLAIN select avg(distinct c2) from t1 group by c2;
QUERY PLAN
-----
--
Gather Motion 3:1 (slice2; segments: 3) (cost=220454.60..255981.36 rows=2001618 width=36)
  -> HashAggregate (cost=220454.60..255981.36 rows=667206 width=36)
        Group By: t1.c2
        -> HashAggregate (cost=146107.39..179923.79 rows=667206 width=8)
              Group By: t1.c2, t1.c2
              -> Redistribute Motion 3:3 (slice1; segments: 3) (cost=37230.32..102282.90 rows=667206
width=8)
                    Hash Key: t1.c2
                    -> HashAggregate (cost=37230.32..62250.54 rows=667206 width=8)
                          Group By: t1.c2, t1.c2
                          -> Seq Scan on t1 (cost=0.00..22218.18 rows=667206 width=4)

Settings: gp_enable_multiphase_agg=on
Optimizer status: legacy query optimizer
(12 rows)
```

Total runtime: 6041.057 ms

Greenplum 中文社区所有

2.8.3 GUC 会影响优化器对多阶段聚集的选择

如果参数 `gp_enable_multiphase_agg` 是打开的，GUC 优化器会选择一个最优的优化器来执行。

多阶段聚集

以下GUC会影响优化器对多阶段聚集的选择。

- `optimizer_force_multistage_agg`: for ORCA
- `gp_enable_multiphase_agg`: for legacy planner

Greenplum 中文社区所有

2.9 分区裁剪

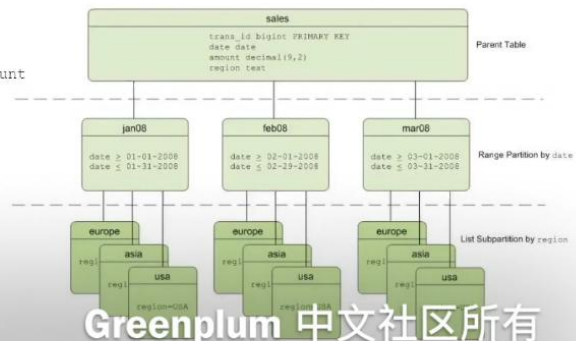
2.9.1 定义分区



分区裁剪

如果有分区表, 判断分区裁剪是否有效。分区裁剪需要查询条件(WHERE 子句)必须和分区条件一样。

```
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 中文社区所有

2.9.2 使用查询计划查看分区

2.9.2.1 planner 分区



分区裁剪

legacy planner:

```
# EXPLAIN SELECT * FROM sales WHERE date='2008-05-07' AND amount=100;
QUERY PLAN
-----
Gather Motion 3:1 (slices: 3) (cost=0.00..2073.00 rows=4 width=53)
-> Append (cost=0.00..2073.00 rows=2 width=53)
   -> Seq Scan on sales_1_prt_5_2_prt_usa sales (cost=0.00..691.00 rows=1 width=53)
       Filter: date = '2008-05-07'::date AND amount = 100::numeric
   -> Seq Scan on sales_1_prt_5_2_prt_asia sales (cost=0.00..691.00 rows=1 width=53)
       Filter: date = '2008-05-07'::date AND amount = 100::numeric
   -> Seq Scan on sales_1_prt_5_2_prt_europe sales (cost=0.00..691.00 rows=1 width=53)
       Filter: date = '2008-05-07'::date AND amount = 100::numeric
Optimizer status: legacy query optimizer
(9 rows)
```

Greenplum 中文社区所有

2.9.2.2 ORCA 分区



分区裁剪

ORCA:

```
# EXPLAIN SELECT * FROM sales WHERE date='05-07-08' AND amount=100;
                                QUERY PLAN
-----
Gather Motion 3:1 (slice1; segments: 3) (cost=0.00..431.00 rows=1 width=24)
->  Sequence (cost=0.00..431.00 rows=1 width=24)
     -> Partition Selector for sales (dynamic scan id: 1) (cost=10.00..100.00 rows=34 width=4)
          Partitions selected: 3 (out of 36)
     -> Dynamic Table Scan on sales (dynamic scan id: 1) (cost=0.00..431.00 rows=1 width=24)
          Filter: date = '2008-05-07'::date AND amount = 100::numeric
Optimizer status: PQO version 3.63.1
(7 rows)
```

Greenplum 中文社区所有

2.9.2.3 最优查询条件

如果在分区上做一个查询的时候，能在查询过滤条件个分区条件在一起则是最优的查询语句。



分区裁剪

legacy planner:

```
# EXPLAIN SELECT * FROM sales WHERE date='05-07-08' AND region='usa';
                                QUERY PLAN
-----
Gather Motion 3:1 (slice1; segments: 3) (cost=0.00..691.00 rows=2 width=53)
->  Append (cost=0.00..691.00 rows=1 width=53)
     -> Seq Scan on sales_1_prt_5_2_prt_usa sales (cost=0.00..691.00 rows=1 width=53)
          Filter: date = '2008-05-07'::date AND region = 'usa'::text
Optimizer status: legacy query optimizer
(5 rows)
```

Greenplum 中文社区所有

分区裁剪

ORCA:

```
# EXPLAIN SELECT * FROM sales WHERE date='05-07-08' AND region='usa';
                                QUERY PLAN
-----
Gather Motion 3:1  (slice1; segments: 3)  (cost=0.00..431.00 rows=1 width=24)
->  Sequence  (cost=0.00..431.00 rows=1 width=24)
    ->  Partition Selector for sales (dynamic scan id: 1)  (cost=10.00..100.00 rows=34 width=4)
        Partitions selected:1 (out of 36)
    ->  Dynamic Table Scan on sales (dynamic scan id: 1)  (cost=0.00..431.00 rows=1 width=24)
        Filter: date = '2008-05-07'::date AND region = 'usa'::text
Optimizer status: PQO version 3.63.1
(7 rows)
```

Greenplum 中文社区所有

3 一些最佳实战

3.1 最佳实践注意点

一些最佳实践

- 合理选择分布键, 尽量保持数据在各个 segments 之间平均分布。
- 对于连接和聚集, 尽量使用分布键
 - 可以减少数据在 segments 之间的传输
- 在查询的结果集中, 尽量避免不必要的行 (SELECT *)
- 尽量避免对数据量大的结果集进行排序
- 索引的使用
 - 尽量谨慎地创建索引
 - 确保创建的索引真正在 workload 中被使用了
 - 验证创建的索引是否提升了查询的性能
 - 尽量避免在会被频繁更新的行上创建索引
 - 在进行大量数据加载时, 可以考虑先删除索引
 - ...

Greenplum 中文社区所有