1.优化上面的实时评估表。进行下列操作。

1. 利用设置实时评估数据库的说明，构建并填充这些表格。

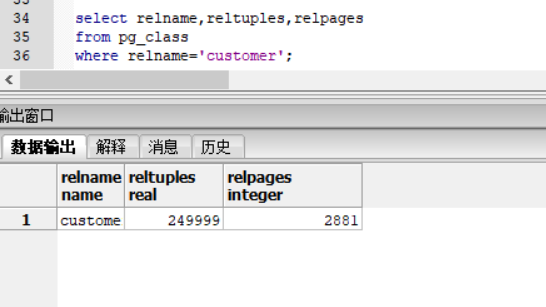
见第五步J

2. 当存储表格到数据库的时候，分析表格的存储特征。你将提交最正确的评估每个表格的元组数量以及用于存储每个表格的磁盘 的页数。 提交收集这些信息的必要代码以及这些查询的输出。对于每个表格， 你以什么为块因子进行计算？

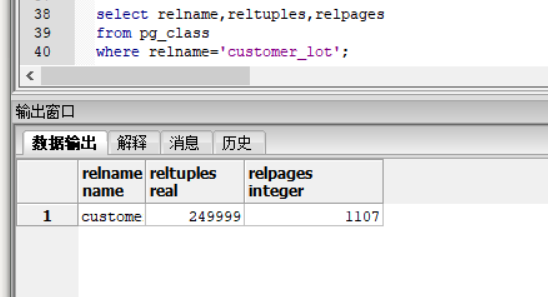
Lot



Customer



Customer\_lot



Lot relation: 249999/2688=93.0056

Customer relation: 249999/1441=173.4899

Customer\_lot relation: 249999/1107=255.8347

3.分析上面给出的每个查询的运行时特征。报告PostgreSQL的评估，用于磁盘访问的实际次数以及每次查询花费时间的总计。 提 交收集这些数据的必要代码。

(1) explain analyze select lot\_id from lot where lot\_size between 3000 and 15000;

"Seq Scan on lot (cost=0.00..6437.98 rows=120071 width=4) (actual time=0.004..23.933 rows=120237 loops=1)"

" Filter: ((lot\_size >= 3000::double precision) AND (lot\_size <= 15000::double precision))"

" Rows Removed by Filter: 129762"

"Total runtime: 25.841 ms"

1. explain analyze select lot\_id from lot where lot\_value between 3000 and 15000;

"Seq Scan on lot (cost=0.00..6437.98 rows=2996 width=4) (actual time=0.010..22.364 rows=3045 loops=1)"

" Filter: ((lot\_value >= 3000::double precision) AND (lot\_value <= 15000::double precision))"

" Rows Removed by Filter: 246954"

"Total runtime: 22.420 ms"

1. explain analyze select \* from customer where customer\_id=2;

"Seq Scan on customer (cost=0.00..6005.99 rows=1 width=16) (actual time=1.148..12.052 rows=1 loops=1)"

" Filter: (customer\_id = 2)"

" Rows Removed by Filter: 249998"

"Total runtime: 12.062 ms"

1. explain analyze insert into customer values (2500,'Vince','Smith');

"Insert on customer (cost=0.00..0.01 rows=1 width=0) (actual time=0.266..0.266 rows=0 loops=1)"

" -> Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.000..0.001 rows=1 loops=1)"

"Total runtime: 0.274 ms"

1. explain analyze delete from customer where customer\_id='250001';

"Delete on customer (cost=0.00..6005.99 rows=1 width=6) (actual time=11.486..11.486 rows=0 loops=1)"

" -> Seq Scan on customer (cost=0.00..6005.99 rows=1 width=6) (actual time=11.484..11.484 rows=0 loops=1)"

" Filter: (customer\_id = 250001)"

" Rows Removed by Filter: 250000"

"Total runtime: 11.501 ms"

1. explain analyze update customer set customer\_first\_name='Vinny' where customer\_id='249001';

"Update on customer (cost=0.00..6005.99 rows=1 width=16) (actual time=11.770..11.770 rows=0 loops=1)"

" -> Seq Scan on customer (cost=0.00..6005.99 rows=1 width=16) (actual time=11.690..11.731 rows=1 loops=1)"

" Filter: (customer\_id = 249001)"

" Rows Removed by Filter: 249999"

"Total runtime: 11.785 ms"

1. explain analyze select avg(lot\_size) from lot;

"Aggregate (cost=5812.99..5813.00 rows=1 width=8) (actual time=24.762..24.763 rows=1 loops=1)"

" -> Seq Scan on lot (cost=0.00..5187.99 rows=249999 width=8) (actual time=0.004..10.125 rows=249999 loops=1)"

"Total runtime: 24.787 ms"

4. 对于上面每一个查询，如果可以，建议使用一个索引以提高查询性能。要完整地回答这个问题，你必须描述出你编入索引的列 以及你将使用的索引。你必须用一个完整的解释来充分地为你的选择进行辩护。如果一个索引不适合这个查询，那么要清晰地 解释原因。注意，你可以只使用PostgreSQL,支持的索引，例如, hash或b-tree。你也可以用聚类。

查询一和二

对特定范围的值进行顺序扫描，可以使用b树索引

查询三

在客户id相等的条件下进行查询，所以可以使用哈希索引

查询四

这是一个插入操作，使用索引会降低效率，所以不使用索引

查询五和六

使用顺序扫描，在customer\_id和lot\_id上添加哈希索引

查询七

聚集函数使用索引效率不高，所以不使用索引

建立索引：

create index customer\_id\_index on customer using hash(customer\_id);

create index lot\_id\_index on lot using hash(lot\_id);

create index lot\_value\_index on lot using btree(lot\_value);

create index lot\_size\_index on lot using btree(lot\_size);

5.实现那些你之前提出的索引并且分析步骤3中运行的那些些查询的运行时性能。提交一个表，显示出使用索引后，评估的和实际总的磁盘页的访问数以及查询运行时间。基于运行时间，计算出由引表格导致的性能的增加或降低的百分比如果性能没有因为索引而得到提高，那么你必须解释原因。你可以用下面的表格作为基础。

建立完索引之后再次执行分析：

1. explain analyze select lot\_id from lot where lot\_size between 3000 and 15000;

"Seq Scan on lot (cost=0.00..6437.98 rows=120071 width=4) (actual time=0.005..23.217 rows=120237 loops=1)"

" Filter: ((lot\_size >= 3000::double precision) AND (lot\_size <= 15000::double precision))"

" Rows Removed by Filter: 129762"

"Total runtime: 25.099 ms"

1. explain analyze select lot\_id from lot where lot\_value between 3000 and 15000;

"Bitmap Heap Scan on lot (cost=67.12..2924.73 rows=2995 width=4) (actual time=0.518..1.884 rows=3045 loops=1)"

" Recheck Cond: ((lot\_value >= 3000::double precision) AND (lot\_value <= 15000::double precision))"

" -> Bitmap Index Scan on lot\_value\_index (cost=0.00..66.37 rows=2995 width=0) (actual time=0.358..0.358 rows=3045 loops=1)"

" Index Cond: ((lot\_value >= 3000::double precision) AND (lot\_value <= 15000::double precision))"

"Total runtime: 1.969 ms"

1. explain analyze select \* from customer where customer\_id=2;

"Index Scan using customer\_id\_index on customer (cost=0.00..8.02 rows=1 width=16) (actual time=0.010..0.010 rows=1 loops=1)"

" Index Cond: (customer\_id = 2)"

"Total runtime: 0.023 ms"

1. explain analyze insert into customer values (2500,'Vince','Smith');

"Insert on customer (cost=0.00..0.01 rows=1 width=0) (actual time=0.274..0.274 rows=0 loops=1)"

" -> Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.000..0.000 rows=1 loops=1)"

"Total runtime: 0.282 ms"

1. explain analyze delete from customer where customer\_id='250001';

"Delete on customer (cost=0.00..8.02 rows=1 width=6) (actual time=0.010..0.010 rows=0 loops=1)"

" -> Index Scan using customer\_id\_index on customer (cost=0.00..8.02 rows=1 width=6) (actual time=0.009..0.009 rows=0 loops=1)"

" Index Cond: (customer\_id = 250001)"

"Total runtime: 0.020 ms"

1. explain analyze update customer set customer\_first\_name='Vinny' where customer\_id='249001';

"Update on customer (cost=0.00..8.02 rows=1 width=16) (actual time=0.022..0.022 rows=0 loops=1)"

" -> Index Scan using customer\_id\_index on customer (cost=0.00..8.02 rows=1 width=16) (actual time=0.010..0.010 rows=1 loops=1)"

" Index Cond: (customer\_id = 249001)"

"Total runtime: 0.034 ms"

1. explain analyze select avg(lot\_size) from lot;

"Aggregate (cost=5812.99..5813.00 rows=1 width=8) (actual time=24.158..24.158 rows=1 loops=1)"

" -> Seq Scan on lot (cost=0.00..5187.99 rows=249999 width=8) (actual time=0.003..9.940 rows=249999 loops=1)"

"Total runtime: 24.178 ms"

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Without Index | | | With Index | | | Performance Improvement |
| Query Number | Estimated Disk Accesses | Actual Disk Accesses | Actual Run Time | Estimated Disk Accesses | Actual Disk Accesses | Actual Run Time |  |
| 1 | 6437.98 | 23.933 | 25.841 | 6437.98 | 23.217 | 25.099 | 2.91% |
| 2 | 6437.98 | 22.364 | 22.420 | 2924.73 | .1.884 | 1.969 | 91.22% |
| 3 | 6005.99 | 12.052 | 12.062 | 8.02 | 0.010 | 0.023 | 99.81% |
| 4 | 0.01 | 0.266 | 0.274 | 0.01 | 0.274 | 0.282 | -2.92% |
| 5 | 6005.99 | 11.486 | 11.501 | 8.02 | 0.010 | 0.020 | 99.83% |
| 6 | 6005.99 | 11.770 | 11.785 | 8.02 | 0.022 | 0.034 | 99.72% |
| 7 | 5813.00 | 24.763 | 24.787 | 5813.00 | 24.158 | 24.178 | 0.04% |

1. 给定一个你已有的数据，索引结构使哪些查询减速了，使哪些查询加速了？如果1,2,3类查询是常见的（75%的发生率），而4类 是不常见的（25%的发生率），一些查询性能的提高是否超过了其他查询所降低的？如果1,2,3类查询的比率接近50%并且4类也 占50%，那么你的想法会发生怎样改变？

从上面的数据可以看出，唯一使用索引之后变慢的查询是查询四插入操作。

针对目前给出的数据库数据来说，凡采用给出索引的查询，性能都有明显的提高。

如果1，2，3类的查询操作寻占比75%，那么使用索引会使查询效率提高很多，我会使用索引

如果第四类的查询占比50%，那么使用索引和不使用索引的效率可能相差不大，在这种情况下我不会使用索引，可以节省系统开销。

2. 为了研究学习，你正设计一个用于存储传感器记录的数据库。第一年，通过插入语句，将从感知器收集到的记录填充到数据库中。第一年以后，数据库主要用于查询通过select语句。你必须决定是否在最初或是等到第一年以后再索引数据库。从你已经证明的内容，你将在最初索引数据库还是等到第一年以后再做？解释。

我会等到一年之后再建立索引。

因为索引会降低插入速度，所以当表变为静态的之后没索引的查询速度会快很多，这样可以拥有更好的性能。

3. 假设你被一个本地医院雇佣作为数据库专家组的一员。一位同事已经在PostgreSQL中实现了一个存储病人信息的表。要求你获取所有女病人的记录。 你写了一个查询select \* from patient where gender='f'；你主要查询性能是比较差的。现在你那位实现病人表的同事不在，你有责任确定哪是错误的。你首先描述了病人的表格。

hospital=# \d patient

Table "patient"

Column | Type | Modifiers

--------------------+--------------+-------------

id | integer | not null

firstname | text | not null

lastname | text | not null

title | text |

admissiondate | date |

address | text |

gender | char | default 'f'

Indexes: patient\_gender,

patient\_id,

patient\_firstname,

patient\_lastname

Primary key: patient\_pkey

You notice that the gender column is indexed and still are unsure why query performance is not good. You decide to run an explain analyze on the above query. The output generated by explain analyze is:

hospital=# explain SELECT \* FROM patient WHERE gender='f';

NOTICE: QUERY PLAN:

Seq Scan on patient (cost=0.00..173.07 rows=6406 width=70)

EXPLAIN

你现在明白了为什么查询性能这么差了。尽管在gender上有一个索引，但它好像并没有被使用。 你给出了一个查询用于统计表格中男病人和女病人的数量。你发现两者的分布是各占50%。 解释一下为什么索引没有被使用。 对patient\_gender进行聚类是否对性能有帮助？请解释。

因为男女性别分布各占百分之五十。即使有索引，DBMS也得扫描每个磁盘块才能找到所有数据，因为每个磁盘块同时包含男女数据的概率很高。

如果索引是聚集的，那么索引可以扫描数据的子集。对patient\_gender进行聚集，可以将男性和女性组合在一起，那么DBMS只需要查询百分之五十的磁盘块，这样就大大提升了查询速度。

1. 回忆一下PostgreSQL将表格统计信息存储到一个名为pg\_class系统表中。查询规划师每次查询都要访问这个表。这些统计只能用分析命令进行更新。如果分析命令不经常运行，那么表中的统计信息或许是不正确的并且查询规划者或许做出一些会降低系统性能的不好的决策。对于一个策略为每一次查询生成这些统计信息（包括查找，增加，更新，删除）。这个方法可以让查询规划师能够得到最新的统计信息。为什么PostgreSQL不能做这项工作？

这样会降低每次查询的性能。因为查询规划是必须为所有查询生成这些统计信息，这样会使每次查询的效率降低。