



链滴

postgresql 数据库单表查询时, 不同数据量 以及查询 sql 分析

作者: [DavinciDevil](#)

原文链接: <https://ld246.com/article/1558437674236>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

最近项目遇到了一个单表查询慢的情况，需要优化。原因是group by 和limit一起用的时候索引不生，我的解决方法是用子查询，这样索引生效了，查询速度也变快了。

于是我又做了一些其他的测试，这里记录一下我测试分析的过程。ps:懒得造数据了，表名我打马赛克了

1.因为查询条件需要create_user_id (有btree索引)，所以先对表的create_user_id进行分组统计条数

```
select create_user_id,count(*) as user_count from mytable group by create_user_id  
ORDER BY user_count desc
```

create_user_id	user_count
50500	209123
52200	20166
(Null)	7756
51800	2301
51550	1331
50250	982

取50500,52200,51800的数据查询

2.直接查询

```
explain (analyze,buffer, COSTS) select * from [redacted] where create_user_id = 50500 order by id limit 100  
窗口  
数据输出 解释 消息 历史  
QUERY PLAN  
text  
1 Limit (cost=0.42..27.35 rows=100 width=302) (actual time=9.826..10.271 rows=100 loops=1)  
2 Buffers: shared hit=3874  
3 -> Index Scan using pk [redacted] on [redacted] (cost=0.42..56374.76 rows=209374 width=302) (actual time=9.823..10.248 rows=100 loops=1)  
4 Filter: (create user id = 50500)  
5 Rows Removed by Filter: 5808  
6 Buffers: shared hit=3874  
7 Planning time: 0.409 ms  
8 Execution time: 10.355 ms  
  
explain (analyze,buffer, COSTS) select * from [redacted] where create_user_id = 52200 order by id limit 100  
<  
数据输出 解释 消息 历史  
QUERY PLAN  
text  
1 Limit (cost=0.42..278.99 rows=100 width=302) (actual time=369.545..369.730 rows=100 loops=1)  
2 Buffers: shared hit=197690  
3 -> Index Scan using pk [redacted] on [redacted] (cost=0.42..56374.76 rows=20237 width=302) (actual time=369.542..369.702 rows=100 loops=1)  
4 Filter: (create user id = 52200)  
5 Rows Removed by Filter: 223108  
6 Buffers: shared hit=197690  
7 Planning time: 0.430 ms  
8 Execution time: 369.817 ms  
  
explain (analyze,buffer, COSTS) select * from [redacted] where create_user_id = 51800 order by id limit 100  
窗口  
数据输出 解释 消息 历史  
QUERY PLAN  
text  
1 Limit (cost=0.42..2672.19 rows=100 width=302) (actual time=5.031..5.443 rows=100 loops=1)  
2 Buffers: shared hit=2393  
3 -> Index Scan using pk [redacted] on [redacted] (cost=0.42..56374.76 rows=2110 width=302) (actual time=5.029..5.413 rows=100 loops=1)  
4 Filter: (create user id = 51800)  
5 Rows Removed by Filter: 2783  
6 Buffers: shared hit=2393  
7 Planning time: 0.403 ms  
8 Execution time: 5.526 ms
```

原始查询都没有用到索引，意料之中

3.子查询

```
explain (analyze, buffers, COSTS) select a.* from ( select * from ... where create_user_id = 50500 ) a order by a.id limit 100
```

窗口

数据输出 解释 消息 历史

QUERY PLAN
text

```
1 Limit (cost=0.42..27.35 rows=100 width=302) (actual time=10.805..11.247 rows=100 loops=1)
2 Buffers: shared hit=3874
3 -> Index Scan using ... on ... (cost=0.42..56374.76 rows=209374 width=302) (actual time=10.802..11.225 rows=100 loops=1)
4 Filter: (create user id = 50500)
5 Rows Removed by Filter: 5808
6 Buffers: shared hit=3874
7 Planning time: 0.540 ms
8 Execution time: 11.336 ms
```

```
explain (analyze, buffers, COSTS) select a.* from ( select * from ... where create_user_id = 52200 ) a order by a.id limit 100
```

窗口

数据输出 解释 消息 历史

QUERY PLAN
text

```
1 Limit (cost=0.42..278.99 rows=100 width=302) (actual time=360.952..361.140 rows=100 loops=1)
2 Buffers: shared hit=197690
3 -> Index Scan using pk ... on ... (cost=0.42..56374.76 rows=20237 width=302) (actual time=360.947..361.107 rows=100 loops=1)
4 Filter: (create user id = 52200)
5 Rows Removed by Filter: 223108
6 Buffers: shared hit=197690
7 Planning time: 0.462 ms
8 Execution time: 361.235 ms
```

```
explain (analyze, buffers, COSTS) select a.* from ( select * from ... where create_user_id = 51800 ) a order by a.id limit 100
```

窗口

数据输出 解释 消息 历史

QUERY PLAN
text

```
1 Limit (cost=0.42..2672.19 rows=100 width=302) (actual time=6.601..7.167 rows=100 loops=1)
2 Buffers: shared hit=2375 read=18
3 -> Index Scan using ... on ... (cost=0.42..56374.76 rows=2110 width=302) (actual time=6.598..7.127 rows=100 loops=1)
4 Filter: (create user id = 51800)
5 Rows Removed by Filter: 2783
6 Buffers: shared hit=2375 read=18
7 Planning time: 0.425 ms
8 Execution time: 7.262 ms
```

关于这个问题我就是用子查询解决的，但是这里却没有用到索引？？于是我对比了我解决问题的sql和这个sql的不同，发现我这里是用了select *，于是

```
explain (analyze, buffers, COSTS) select * from ( select id as magid from ... where create_user_id = 52200 ) a order by a.magid limit 100
```

窗口

数据输出 解释 消息 历史

QUERY PLAN
text

```
1 Limit (cost=0.42..278.99 rows=100 width=8) (actual time=382.565..382.742 rows=100 loops=1)
2 Buffers: shared hit=197690
3 -> Index Scan using pk ... on ... (cost=0.42..56374.76 rows=20237 width=8) (actual time=382.560..382.703 rows=100 loops=1)
4 Filter: (create user id = 52200)
5 Rows Removed by Filter: 223108
6 Buffers: shared hit=197690
7 Planning time: 0.350 ms
8 Execution time: 382.796 ms
```

这里还是没有用到索引？郁闷了一会，在对比一下。发现原来的是 select id || " 而不是select id，是

```
explain (analyze, buffers, COSTS) select * from ( select id || '' as magid from ... where create_user_id = 52200 ) a order by a.magid limit 100
```

窗口

数据输出 解释 消息 历史

QUERY PLAN
text

```
1 Limit (cost=14230.18..14230.43 rows=100 width=32) (actual time=48.283..48.332 rows=100 loops=1)
2 Buffers: shared hit=901
3 -> Sort (cost=14230.18..14280.78 rows=20237 width=32) (actual time=48.281..48.305 rows=100 loops=1)
4 Sort Key: (((simple sms phone number.id)::text || ''::text))
5 Sort Method: top-N heapsort Memory: 29kB
6 Buffers: shared hit=901
7 -> Bitmap Heap Scan on ... (cost=605.26..13456.74 rows=20237 width=32) (actual time=4.971..26.565 rows=20166 loops=1)
8 Recheck Cond: (create user id = 52200)
9 Heap Blocks: exact=789
10 Buffers: shared hit=901
11 -> Bitmap Index Scan on ... index (cost=0.00..600.20 rows=20237 width=0) (actual time=4.719..4.719 rows=20166 loops=1)
12 Index Cond: (create user id = 52200)
13 Buffers: shared hit=112
14 Planning time: 0.425 ms
15 Execution time: 48.401 ms
```

这里果然是用到了索引，但是我不明白为什么“select id || ” 会用索引，但是“select id”不用索引，网上资料说是pg在查询时有自己一套策略，会根据当前sql情况决定执行方式。但是我不懂这里用了索引更快了才对，而且我拼接空字符串应该也要额外消耗吧。可是结果反而是更快了，

lushed。

继续测试，还是这个sql，id换成50500

```
explain (analyze, buffers, COSTS) select * from ( select id || '' as msgid from   
 where create_user_id = 50500 ) a order by a.msgid limit 100
```

窗口

数据输出 解释 消息 历史

```
QUERY PLAN  
text  
1 Limit (cost=24645.30..24645.55 rows=100 width=32) (actual time=512.544..512.598 rows=100 loops=1)  
2 Buffers: shared hit=12030  
3 -> Sort (cost=24645.30..25168.74 rows=209374 width=32) (actual time=512.541..512.567 rows=100 loops=1)  
4 Sort Key: ((simple sms phone number.id)::text || ''::text)  
5 Sort Method: top-N heapsort Memory: 29kB  
6 Buffers: shared hit=12030  
7 -> Seq Scan on (cost=0.00..16643.18 rows=209374 width=32) (actual time=0.054..353.224 rows=209123 loops=1)  
8 Filter: (create user id = 50500)  
9 Rows Removed by Filter: 34308  
0 Buffers: shared hit=12030  
1 Planning time: 0.454 ms  
2 Execution time: 512.657 ms
```

可以看到不使用索引了，结合我今天查到的资料，应该是pg认为这次的查询返回的结果集较多（配合一张图看），因此不使用索引。

id用51800替换再试试

```
explain (analyze, buffers, COSTS) select * from ( select id || '' as msgid from   
 where create_user_id = 51800 ) a order by a.msgid limit 100
```

窗口

数据输出 解释 消息 历史

```
QUERY PLAN  
text  
1 Limit (cost=5544.15..5544.40 rows=100 width=32) (actual time=7.312..7.360 rows=100 loops=1)  
2 Buffers: shared hit=379 read=9  
3 -> Sort (cost=5544.15..5549.42 rows=2110 width=32) (actual time=7.310..7.333 rows=100 loops=1)  
4 Sort Key: (   
 ::text || ''::text)  
5 Sort Method: top-N heapsort Memory: 29kB  
6 Buffers: shared hit=379 read=9  
7 -> Index Scan using index on simple sms phone number (cost=0.42..5463.50 rows=2110 width=32) (actual time=0.111..3.833 rows=2301 loops=1)  
8 Index Cond: (create user id = 51800)  
9 Buffers: shared hit=379 read=9  
0 Planning time: 0.442 ms  
1 Execution time: 7.423 ms
```

有使用索引，和预期的一样

4.用with语句

```
explain (analyze, buffers, COSTS) with cte as ( select id from   
 where create_user_id = 50500 ) select * from cte order by id limit 100
```

窗口

数据输出 解释 消息 历史

```
QUERY PLAN  
text  
1 Limit (cost=27262.48..27262.73 rows=100 width=8) (actual time=450.770..450.832 rows=100 loops=1)  
2 Buffers: shared hit=12030, temp written=459  
3 CTE cte  
4 -> Seq Scan on (cost=0.00..15072.88 rows=209374 width=8) (actual time=0.037..237.247 rows=209123 loops=1)  
5 Filter: (create user id = 50500)  
6 Rows Removed by Filter: 34308  
7 Buffers: shared hit=12030  
8 -> Sort (cost=12189.60..12713.04 rows=209374 width=8) (actual time=450.768..450.801 rows=100 loops=1)  
9 Sort Key: cte.id  
0 Sort Method: top-N heapsort Memory: 29kB  
1 Buffers: shared hit=12030, temp written=459  
2 -> CTE Scan on cte (cost=0.00..4187.48 rows=209374 width=8) (actual time=0.042..392.124 rows=209123 loops=1)  
3 Buffers: shared hit=12030, temp written=459  
4 Planning time: 0.351 ms  
5 Execution time: 453.481 ms
```

```

explain (analyze, buffers, COSTS) with cte as ( select id from [redacted] where create_user_id = 52200 ) select * from cte order by id limit 100

```

窗口

数据输出 解释 消息 历史

QUERY PLAN
text

```

1 Limit (cost=14483.15..14483.40 rows=100 width=8) (actual time=31.168..31.213 rows=100 loops=1)
2   Buffers: shared hit=901
3   CTE cte
4     -> Bitmap Heap Scan on simple sms phone number (cost=605.26..13304.96 rows=20237 width=8) (actual time=4.095..13.293 rows=20166 loops=1)
5         Recheck Cond: (create user id = 52200)
6         Heap Blocks: exact=789
7         Buffers: shared hit=901
8     -> Bitmap Index Scan on [redacted] id index (cost=0.00..600.20 rows=20237 width=0) (actual time=3.860..3.860 rows=20166 loops=1)
9         Index Cond: (create user id = 52200)
10        Buffers: shared hit=112
11 -> Sort (cost=1178.18..1228.78 rows=20237 width=8) (actual time=31.167..31.188 rows=100 loops=1)
12     Sort Key: cte.id
13     Sort Method: top-N heapsort Memory: 29kB
14     Buffers: shared hit=901
15 -> CTE Scan on cte (cost=0.00..404.74 rows=20237 width=8) (actual time=4.100..25.484 rows=20166 loops=1)
16     Buffers: shared hit=901
17 Planning time: 0.321 ms
18 Execution time: 31.621 ms

```

```

explain (analyze, buffers, COSTS) with cte as ( select id from [redacted] where create_user_id = 51800 ) select * from cte order by id limit 100

```

窗口

数据输出 解释 消息 历史

QUERY PLAN
text

```

1 Limit (cost=5570.52..5570.77 rows=100 width=8) (actual time=2.482..2.504 rows=100 loops=1)
2   Buffers: shared hit=388
3   CTE cte
4     -> Index Scan using simple [redacted] index on simple sms phone number (cost=0.42..5447.68 rows=2110 width=8) (actual time=0.032..1.302 rows=2110 loops=1)
5         Index Cond: (create user id = 51800)
6         Buffers: shared hit=388
7     -> Sort (cost=122.84..128.12 rows=2110 width=8) (actual time=2.480..2.489 rows=100 loops=1)
8         Sort Key: cte.id
9         Sort Method: top-N heapsort Memory: 29kB
10        Buffers: shared hit=388
11 -> CTE Scan on cte (cost=0.00..42.20 rows=2110 width=8) (actual time=0.035..2.107 rows=2301 loops=1)
12     Buffers: shared hit=388
13 Planning time: 0.207 ms
14 Execution time: 2.566 ms

```

这里的预期结果和子查询差不多，不同在于with比子查询在查询同样条件下会快一些。还有就是with查询不能用“select id ||”，会报错。

最后我的结论是：

- 1.pg的group by 和limit 共用的时候确实会导致索引不生效，可以用子查询或者with语句解决
- 2.pg的sql执行策略决定是否使用索引的因素有很多，比如返回结果集占该表百分比的大小（试过在他数据库数测试测试返回结果集也是20多万但是占总表百分比没这么大，也用了索引），select部分的写法（这里我并不理解为什么）以及其他。