增加索引真能提升查询性能吗？

翻译：陈雁飞 校对：李冉

### 作者简介

[**Laurenz Albe**](https://www.cybertec-postgresql.com/en/author/cybertec_albe/) **:** Cybertec公司的高级顾问和支持工程师。13年来，他一直参与并给PostgreSQL数据库做出贡献。

### 译者简介

**陈雁飞**，开源PostgreSQL爱好者，一直从事PostgreSQL数据库运维工作

**李冉，**瀚高基础软件工具开发工程师。

众所周知，当创建一个新的索引之后，数据的修改操作会变慢并且该索引会占据一定的磁盘存储空间，这是我们创建一个新索引必须付出的代价。这也是为什么你需要尝试[没有创建多余索引的原因](https://www.cybertec-postgresql.com/en/get-rid-of-your-unused-indexes/)。

但是很多人认为SELECT查询语句不会受到一个新索引的影响。当心索引没有使用到的时候，就会发生更糟糕的情况。

当然，这个并不总是正确的。我在实际中看到不止一个这样的例子。下面我将给你介绍一个这样的案例，并告诉你能做些什么。

### 一个例子

使用下面的表进行测试

|  |
| --- |
| CREATE TABLE skewed (     sort        integer NOT NULL,     category    integer NOT NULL,     interesting boolean NOT NULL  );    INSERT INTO skewed     SELECT i, i%1000, i>50000     FROM generate\_series(1, 1000000) i;    CREATE INDEX skewed\_category\_idx ON skewed (category);    VACUUM (ANALYZE) skewed; |

我们希望找到前面20个且category值为42，insteresting值为真的元组。

|  |
| --- |
| EXPLAIN (ANALYZE, BUFFERS)  SELECT \* FROM skewed  WHERE interesting AND category = 42  ORDER BY sort  LIMIT 20; |

这个查询语句的性能很好

|  |
| --- |
| QUERY PLAN  --------------------------------------------------------------------   Limit  (cost=2528.75..2528.80 rows=20 width=9)          (actual time=4.548..4.558 rows=20 loops=1)     Buffers: shared hit=1000 read=6     ->  Sort  (cost=2528.75..2531.05 rows=919 width=9)               (actual time=4.545..4.549 rows=20 loops=1)           Sort Key: sort           Sort Method: top-N heapsort  Memory: 25kB           Buffers: shared hit=1000 read=6           ->  Bitmap Heap Scan on skewed                          (cost=19.91..2504.30 rows=919 width=9)                          (actual time=0.685..4.108 rows=950 loops=1)                 Recheck Cond: (category = 42)                 Filter: interesting                 Rows Removed by Filter: 50                 Heap Blocks: exact=1000                 Buffers: shared hit=1000 read=6                 ->  Bitmap Index Scan on skewed\_category\_idx                          (cost=0.00..19.68 rows=967 width=0)                          (actual time=0.368..0.368 rows=1000 loops=1)                       Index Cond: (category = 42)                       Buffers: shared read=6   Planning time: 0.371 ms   Execution time: 4.625 ms |

PostgreSQL使用索引扫描方法找到category值为42的1000个元组，筛选掉一些不感兴趣的，然后对其排序并返回前面20个元组。5毫秒的执行时间看起来非常不错。

### 一个新索引让事情变得糟糕

下面我们将增加一个索引，因为索引对排序有帮助。如果我们要找到前20个结果，那么绝对将是有意思的结果：

|  |
| --- |
| CREATE INDEX skewed\_sort\_idx ON skewed (sort); |

突然间，情况变得更糟糕：

|  |
| --- |
| QUERY PLAN  -------------------------------------------------------------   Limit  (cost=0.42..736.34 rows=20 width=9)          (actual time=21.658..28.568 rows=20 loops=1)     Buffers: shared hit=374 read=191     ->  Index Scan using skewed\_sort\_idx on skewed                  (cost=0.42..33889.43 rows=921 width=9)                  (actual time=21.655..28.555 rows=20 loops=1)           Filter: (interesting AND (category = 42))           Rows Removed by Filter: 69022           Buffers: shared hit=374 read=191   Planning time: 0.507 ms   Execution time: 28.632 ms |

**发生了什么？**

PostgreSQL认为利用索引顺序查找可以更加快速的匹配到满足条件的20个元组。但是数据库并不知道满足条件的行是如何按照排序列分布的，也就是说数据库并不知道需要扫描69042行才能找到20个匹配元组（执行计划中显示过滤了69022行）。

### 我们该怎样做来获得更好的计划？

PostgreSQL 10版本中增加了[扩展统计信息](https://www.postgresql.org/docs/current/static/planner-stats.html#PLANNER-STATS-EXTENDED)功能，用于跟踪记录不同列之间的相关性，但是这不会记录值的分布情况，因此对我们没有帮助。

有两种解决方法：

1. **删除误导PostgreSQL的索引**。如果可行，这是一个简单的解决方案。但是通常情况下不能这样做，因为索引要么用于保证唯一性约束，或者可以用于加速其他查询语句。
2. **重写查询语句，以便PostgreSQL不使用不当的索引。**在许多可能的解决方案中，我列举下面两个：

* **带有offset 0的子查询**

|  |
| --- |
| SELECT \*  FROM (SELECT \* FROM skewed  WHERE interesting AND category = 42  OFFSET 0) q  ORDER BY sort  LIMIT 20; |

这里利用OFFSET和LIMIT子句阻止进行上提子查询的优化，即使它们对查询结果没有任何影响。

* 使用表达式作为排序键

|  |
| --- |
| SELECT \* FROM skewed  WHERE interesting AND category = 42  ORDER BY sort + 0  LIMIT 20; |

这里利用了PostgreSQL无法推断sort+0与sort相同的特性。请记住，PostgreSQL是支持扩展的，你可以定义自己的+运算符！

原文地址：<https://www.cybertec-postgresql.com/en/index-decreases-select-performance/>

### 译后感

在上面的例子中，增加索引之后导致优化器选择非最优的执行计划本原因在于：目前优化器假设数据在页面中是均匀分布的，但是实际情况却不一定，因此导致代价估计产生“偏差”。详细分析可以参考德哥的博客 《[PostgreSQL 优化器案例之 - order by limit 索引选择问题](https://github.com/digoal/blog/blob/master/201807/20180712_01.md)》，可以计算出一个“分水岭”值。