去年十二月份，VWO 平台支持团队发布了一份缺陷报告。这份报告很有意思，其中有一个来自某家企业用户的分析查询，它的运行速度非常慢。因为我是这个数据平台的一员，所以立马开始着手诊断这个问题。

背 景

首先，我觉得有必要介绍一下 VWO（https://vwo.com/）平台。人们可以在这个平台上运行各种与他们的网站有关的工作负载，比如 A/B 测试、跟踪访问用户、转换、漏斗分析、渲染热点图、重放访问用户步骤，等等。

这个平台真正强大的地方在于它所提供的报告。如果没有这个平台，即使企业用户收集了大量数据也是毫无用处的，因为他们无法从数据中获取洞见。

有了这个平台，用户可以针对海量数据执行各种复杂的查询，比如下面这个：

复制代码

```
Showall clicks by visitors on webpage "abc.com"
FROM <date d1> TO <date d2>
for people who were either
using Chrome as a browser OR
(were browsing from Europe AND were using iPhone devices)
```

请注意查询中的布尔运算符，查询接口为用户提供了这些东西，他们可以随意运行复杂的查询来获得想要的数据。

慢查询

这个用户执行的查询从表面上看应该是很快的：

复制代码

```
Show me all session recordings
for users who visited any webpage
containing the url that matches the pattern "/jobs"
```

这个网站的流量是非常巨大的，我们保存了数百万个唯一的 URL。这个用户想要查询符合他们业务需求的 URL。

初步诊断

现在让我们来看看在数据库方面都发生了什么。下面是相应的 SQL 语句：

复制代码

```
SELECT
count(*)
FROM
acc_{account_id}.urls as recordings_urls,
acc_{account_id}.recording_data as recording_data,
acc_{account_id}.sessions as sessions
WHERE
recording_data.usp_id = sessions.usp_id
AND sessions.referrer_id = recordings_urls.id
AND ( urls && array(selectidfrom acc_{account_id}.urls whereurlILIKE'%enterprise_customer.
AND r_time > to_timestamp(1542585600)
AND r_time < to_timestamp(1545177599)
AND recording_data.duration >=5
AND recording_data.num_of_pages > 0 ;
```

这是它的执行时间：

复制代码

```
Planning time: 1.480ms
Execution time: 1431924.650ms
```

这个语句查询的行数在 15 万行左右。查询计划显示了一些信息，但还不能看出瓶颈出在哪里。

我们再来进一步分析一下查询语句。这个语句连接了三张表：

1. sessions：用于展示会话信息的表，例如 browser、user-agent、country，等等。
2. recording_data：记录 url、页面、时间段，等等。
3. urls：为了避免出现重复的 url，我们使用单独的表对 url 进行了规范化。

另外请注意，我们使用 account_id 对这三表进行了分区，所以不可能出现因为某些账号记录过多导致性能变慢的情况。

寻找线索

经过进一步排查，我们发现这个查询有一些不一样的地方。比如下面这行：

复制代码

```
urls && array(
selectidfrom acc_{account_id}.urls
whereurlILIKE'%enterprise_customer.com/jobs%'
)::text[]
```

最开始我认为针对所有长 URL 执行“ILIKE”操作是导致速度变慢的元凶，但其实并不是！

复制代码

```
SELECT id FROM urls WHERE url ILIKE '%enterprise_customer.com/jobs%';
id
--------
...
```

	(198661 rows)
	Time: 5231.765 ms

模式匹配查询本身只花了 5 秒钟，所以要匹配数百万个 URL 显然并不是个问题。

第二个可疑的地方是 JOIN 语句，或许是大量的连接操作导致速度变慢？一般来说，如果查询速度变慢，我们首先会认为连接操作是罪魁祸首，但对于目前这个情况，我不认为是这样的。

复制代码

	analytics_db=# SELECT
	count(*)
	FROM
	acc_{account_id}.urls as recordings_urls,
	acc_{account_id}.recording_data_0 as recording_data,
	acc_{account_id}.sessions_0 as sessions
	WHERE
	recording_data.usp_id = sessions.usp_id
	AND sessions.referrer_id = recordings_urls.id
	AND r_time > to_timestamp(1542585600)
	AND r_time < to_timestamp(1545177599)
	AND recording_data.duration >=5
	AND recording_data.num_of_pages > 0 ;
	count

	8086
	(1row)
	Time: 147.851 ms

看，JOIN 操作实际上是很快的。

缩小可疑范围

我开始调整查询语句，尽一切可能提升查询性能。我和我的团队想出了两个方案。

针对子查询使用 EXISTS：我们想要进一步确认问题是不是出在 URL 子查询上。一种方法是使用 EXISTS，它会在找到第一条匹配记录时就返回，对性能提升很有帮助。

复制代码

```
SELECT
count(*)
FROM
acc_{account_id}.urls as recordings_urls,
acc_{account_id}.recording_data as recording_data,
acc_{account_id}.sessions as sessions
WHERE
recording_data.usp_id = sessions.usp_id
AND ( 1 = 1 )
AND sessions.referrer_id = recordings_urls.id
AND (exists(select id from acc_{account_id}.urls where url ILIKE'%enterprise_customer.com/jo
AND r_time > to_timestamp(1547585600)
AND r_time < to_timestamp(1549177599)
AND recording_data.duration >=5
AND recording_data.num_of_pages > 0 ;
count
32519
(1row)
Time: 1636.637 ms
```

使用了 EXISTS 后，速度变快了很多。那么问题来了，为什么 JOIN 查询和子查询都很快，但放在一起就变得这么慢呢？

将子查询移到 CTE 中：如果子查询本身很快，我们可以预先计算结果，然后再传给主查询。

□复制代码

```
WITH matching_urls AS (
selectid::textfrom acc_{account_id}.urls whereurlILIKE'%enterprise_customer.com/jobs%'
)

SELECT
count(*) FROM acc_{account_id}.urls as recordings_urls,
acc_{account_id}.recording_data as recording_data,
acc_{account_id}.sessions as sessions,
matching_urls
WHERE
recording_data.usp_id = sessions.usp_id
AND ( 1 = 1 )
AND sessions.referrer_id = recordings_urls.id
AND (urls && array(SELECTidfrom matching_urls)::text[])
AND r_time > to_timestamp(1542585600)
AND r_time < to_timestamp(1545107599)
AND recording_data.duration >=5
AND recording_data.num_of_pages > 0;
```

但这样仍然很慢。

寻找元凶

还有个地方之前一直被我忽略了，但因为没有其他办法了，所以我决定看看这个地方，那就是 && 运算符。既然 EXISTS 对性能提升起到了很大作用，那么剩下的就只有 && 可能会导致查询变慢了。

&& 被用来找出两个数组的公共元素。

初始查询中的 && 是这样的：

□复制代码

```
AND ( urls && array(select id from acc_{account_id}.urls where url ILIKE'%enterprise_custome
```

我们对 URL 进行了模式匹配，然后与所有 URL 进行交集操作。这里的“urls“并不是指包含了所有 URL 的表，而是 recording_data 的”urls“列。

因为现在对 && 有所怀疑，我使用 EXPLAIN ANALYZE 对查询语句进行了分析。

复制代码

	Filter: ((urls && ($0)::text[]) AND (r_time > '2018-12-17 12:17:23+00'::timestampwith time zo
	Rows Removed byFilter: 52710

因为有好多行 &&，说明它被执行了好几次。

我通过单独执行这些过滤条件确认了是这个问题。

复制代码

	SELECT1
	FROM
	acc_{account_id}.urls as recordings_urls,
	acc_{account_id}.recording_data_30 as recording_data_30,
	acc_{account_id}.sessions_30 as sessions_30
	WHERE
	urls && array(selectidfrom acc_{account_id}.urls whereurlILIKE'%enterprise_customer.com/jo

这个查询的 JOIN 很快，子查询也很快，所以问题出在 && 上面。

解决方案

&& 之所以很慢，是因为两个集合都很大。如果我把 urls 替换成{“http://google.com/”，“http://wingify.com/”}，这个操作就很快。

我开始在谷歌上搜索如何在 Postgre 中不使用 && 进行交集操作，但并没有找到答案。

最后，我们决定这样做：获取所有匹配的 urls 行，像下面这样：

复制代码

```
SELECT urls.url
FROM
acc_{account_id}.urls as urls,
(SELECTunnest(recording_data.urls) ASid) AS unrolled_urls
WHERE
urls.id = unrolled_urls.id AND
urls.url ILIKE'%jobs%'
```

这里没有使用 JOIN 语句，而是使用了一个子查询，并展开 recording_data.urls 数组，这样就可以直接在 where 语句中应用查询条件。

这里的 && 用来判断一个给定的 recording 是否包含匹配的 URL。它会遍历数组（或者说表中的行），在条件满足时立即停止，这个看起来是不是跟 EXISTS 很像？

因为我们可以在子查询之外引用 recording_data.urls，在必要时可以使用 EXISTS 来包装子查询。

把所有的东西放在一起，我们就得到了最终这个优化的查询：

复制代码

```
SELECT
count(*)
FROM
acc_{account_id}.urls as recordings_urls,
acc_{account_id}.recording_data as recording_data,
```

```
acc_{account_id}.sessions as sessions
WHERE
recording_data.usp_id = sessions.usp_id
AND ( 1 = 1 )
AND sessions.referrer_id = recordings_urls.id
AND r_time > to_timestamp(1542585600)
AND r_time < to_timestamp(1545177599)
AND recording_data.duration >=5
AND recording_data.num_of_pages > 0
ANDEXISTS(
SELECT urls.url
FROM
acc_{account_id}.urls as urls,
(SELECT unnest(urls) AS rec_url_id FROM acc_{account_id}.recording_data)
AS unrolled_urls
WHERE
urls.id = unrolled_urls.rec_url_id AND
urls.url ILIKE'%enterprise_customer.com/jobs%'
);
```

这个查询的执行时间为 1898.717 毫秒，是不是值得庆祝一下？

等等，我们还要验证一下结果是不是对的。我对 EXISTS 有所怀疑，因为它有可能会改变查询逻辑，导致过早地退出。我们要确保不会在查询中引入新的 bug。

我们对慢查询和快查询结果进行了 count(*) 比较，不同数据集的查询结果都是一致的。对于一些较小的数据集，我们还手动比对了具体数据，也没有问题。

学到的教训

在这次性能排查过程中，我们学到了这些东西：

1. 查询计划并不会告诉我们所有东西，但还是很有用的；
2. 越是明显的疑点越不太可能是元凶；
3. 一个慢查询可能包含多个单独的瓶颈点；

4. 并非所有优化都是可简化的；
5. 在可能的地方使用 EXISTS 来获得大幅性能提升。

结论

我们将一个查询的运行时间从 24 分钟降到了 2 秒钟，一个不可思议的性能提升！我们花了 1 个半到 2 个小时的时间来优化和测试这个查询。SQL 其实是一门非常神奇的语言，只要你放开心态去拥抱它。

英文原文：

https://parallelthoughts.xyz/2019/05/a-tale-of-query-optimization/