


数据库 性能优化案例分析之二：时间区域查询的性能优化

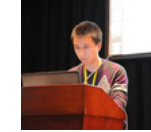
- [社区](#)
- [招聘](#)
- [Wiki](#)
- [酷站](#)
- [Gems](#)

搜索

- [Q](#)
- [发布新话题](#)
- 
- [c332030 你收藏的话题](#)
- [个人设置](#)
- [记事本](#)
- [退出](#)

数据库 性能优化案例分析之二：时间区域查询的性能优化

[guakewang](#) · 2017年11月14日 · 最后由 [iamzhuchunbo](#) 回复于 2019年09月18日 · 16030 次阅读



本帖已被管理员设置为精华贴

目录

[背景](#)[分析过程](#)[第一步：EXPLAIN](#)[第二步：理解执行计划](#)[第三步：分析数据解决方案扩展](#)

背景

我们在项目中经常会遇到把事件显示在日历上的需求，在项目管理，事件追踪，预约等相关功能中很常见，界面通常类似这样：

Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	31	1	2	3	4
			All Day Event			
5	6	7	8	9	10	11
		Long Event	+1 more	+7 more	7a Birthday Ps	
12	13	14	15	16	17	18
				4p Repeating I		
19	20	21	22	23	24	25

常见的做法是给对应表添加开始时间和结束时间的栏位：start_time and end_time，然后构建一个查询得到这个时间段内的数据，示意如下：

|---给定时间段---|
|---A---| |---B---| |-C-| |---D---|

查询要排除掉 A 和 D 这种数据，也就是结束时间小于给定的开始时间，或者开始时间大于给定的结束时间：not (end_time <= :start or start_time >= :end)

从 NOT (A OR B) 可以得到 (NOT A) AND (NOT B)，最终的查询语句是这样：

```
select * from events where end_time > :start and start_time < :end
```

考虑到这是一个范围查找，通常都会给这 2 个栏位加上索引，数据量小的时候，这个查询效率很高，但在数据增加到百万级别之后，在慢查询日志里面经常会发现这个查询耗时超过几百毫秒，随着数据量的增加，甚至超过 3000 毫秒。

分析过程

继续用慢查询的分析过程套路

第一步：EXPLAIN

有时候相同的查询语句，只是查询参数少许不同，会导致预估的 IO Cost 不同，执行计划也会改变，我们可以看到，这个查询有时候会用 start_time 的索引，有时候会用 end_time 的索引：

```
explain select * from events where end_time > '2017-05-01 00:00:00' and start_time < '2017-06-01 00:00:00'\G
```

```
select_type: SIMPLE
table: events
partitions: NULL
type: range
possible_keys: index_events_on_start_time,index_events_on_end_time
key: index_events_on_start_time
key_len: 5
ref: NULL
rows: 799704
filtered: 50.00
Extra: Using index condition; Using where; Using MRR
```

```
explain select * from events where end_time > '2017-08-01 14:00:00' and start_time < '2017-09-01 00:00:00'\G
```

```
select_type: SIMPLE
table: events
partitions: NULL
type: range
possible_keys: index_events_on_start_time,index_events_on_end_time
key: index_events_on_end_time
key_len: 5
ref: NULL
rows: 946430
filtered: 50.00
Extra: Using index condition; Using where; Using MRR
```

第二步：理解执行计划

从执行计划其实已经可以看到这个慢查询的原因：需要检查的行数太多了，预估在 79 万和 94 万。但保险起见，还是从INFORMATION_SCHEMA.OPTIMIZER_TRACE表中获取信息来确认我们的猜测：

```
{
  "index": "index_events_on_start_time",
  "ranges": [
    "start_time < 0x999d02f000"
  ],
  "rows": 3147211,
  "cost": 2.73e6,
  "chosen": false
},
{
  "index": "index_events_on_end_time",
  "ranges": [
    "0x999d02e000 < end_time"
  ],
  "rows": 946430,
  "cost": 826083,
  "chosen": true
}
```

这个查询相当于构建了 无穷小 <= start_time <= ? 和 ? <= end_time <= 无穷大 这样 2 个开放区间查询。在这里由于前者的开放区间内有超过 300 万的数据需要检查，所以执行计划选择了后者相对较小的 94 万作为索引。

第三步：分析数据

这个表有超过 600 万的数据：

```
show table status like 'events'\G

      Name: events
    Engine: InnoDB
     Rows: 6294423
```

所有的数据开始时间和结束时间均匀分布，索引也没有问题，而且查询除了按月，也有按其他维度（天，日，小时，指定范围）来进行查询，通过添加辅助字段也无法减少开放区间的查询数据量，好像进入了一个死胡同...

解决方案

遇到现有知识无法解决的问题，那就 Google 吧，经过尝试不同的关键字（关于如何选择关键字解决日常搬砖遇到的问题，可以留个坑，以后再写）发现了一篇很详细的文章，和我们遇到的情况是一模一样：<https://explainextended.com/2009/07/01/overlapping-ranges-mysql/>

文章提到了将时间区间变成二维线段，然后利用 R-Tree (spatial) 索引，通过范围重叠来解决这个性能问题。看起来很不错，那就开动吧。

添加一个辅助字段并添加索引：

```
alter table events add column time_range linestring;
create spatial index index_events_on_time_range on events(time_range);
```

然后把时间区间转换成线段更新到这个字段：

```
update events set time_range = LineString(Point(-1, UNIX_TIMESTAMP(start_time)), Point(1,UNIX_TIMESTAMP(end_time)));
```

用 MBRIntersects 函数来查询重叠：

```
select * from events where MBRIntersects(time_range, LineString(Point(0, UNIX_TIMESTAMP('2017-08-01 00:00:00')), Point(0, UNIX_TIMESTAMP('2017-09-01 00:00:00'))));
```

80 毫秒! 相同的查询之前需要 2410 毫秒：

```
select * from events where end_time >= '2017-07-01 00:00:00' and start_time <= '2017-08-01 00:00:00';
```

给数据库再加个 trigger，当更新和插入数据时，自动更新 time_range 这个字段：

```
CREATE TRIGGER events_insert_time_range BEFORE INSERT ON events
FOR EACH ROW
BEGIN
    SET NEW.time_range = LineString(Point(-1, UNIX_TIMESTAMP(NEW.start_time)), Point(1, UNIX_TIMESTAMP(NEW.end_time)));
END

CREATE TRIGGER events_update_time_range BEFORE UPDATE ON events
FOR EACH ROW
BEGIN
    IF ((NEW.start_time != OLD.start_time) OR (NEW.end_time != OLD.end_time)) THEN
        SET NEW.date_range = LineString(Point(-1, UNIX_TIMESTAMP(NEW.start_time)), Point(1, UNIX_TIMESTAMP(NEW.end_time)));
    END IF;
END
```

问题解决收工！

扩展

这个问题有很多可以扩展的问题：

我们之前只会将 R-Tree 用在地理位置的查询，从这个问题可以扩展到时间，IP 地址等其他类似的查询，除了这些范围查询还有其他适合应用的场景吗？

文章中构建的线段是 (-1,start_time) 到 (1,end_time) 的一条斜线，用位于 X 等于 0 的线段去查询重叠。为什么不能用更直观的方式（类似文章开头的线段示意图），构建一个 (start_time, 0) 到 (end_time,0) 一条线段，然后用 Y 等于 0 的线段去查询重叠呢？

如果不用数据库 trigger，在 rails 里面有什么方法能够自动更新这个辅助字段呢？甚至更进一步，是否可以删除掉 start_time 和 end_time 这 2 个字段，只用 time_range 这一个字段来满足所有的查询需求呢？

下篇占坑 性能优化案例分析之三：N:M 的用户动态优化

[72 个赞](#)



[numbocoder](#) #1 [2017年11月14日](#)

赞！以前从没用过 R-Tree 的索引，居然还可以这样用，看来以后优化区间查询的，都可以用 R-Tree 来试一下



[rubychinatest](#) #2 [2017年11月14日](#)