

活动中台系统慢 SQL 治理实践

原创 互联网服务器团队 vivo互联网技术 2025年04月02日 20:02 广东

作者：vivo 互联网服务器团队- Zhang Mengtao

活动中台系统作为中台项目非常注重系统性能和用户体验，数据库系统性能问题会对应用程序的性能和用户体验产生负面影响。慢查询可能导致应用程序响应变慢、请求堆积、系统负载增加等问题，甚至引发系统崩溃或不可用的情况，因此，需要在数据库系统中针对执行缓慢的 SQL 查询进行优化和改进。本文主要介绍活动中台系统针对慢SQL问题的实践治理案例。

一、慢 SQL 的含义

1.1 慢 SQL 的含义

慢SQL是指执行时间较长的SQL查询或操作。真实的慢 SQL 通常会伴随着大量的行扫描、临时文件排序或者频繁的磁盘 flush，直接影响就是磁盘 IO 升高，让正常的 SQL 变成了慢 SQL，大面积执行超时。

大家不要被慢查询这个名字误导，以为慢查询日志只会记录 select 语句，其实也会记录执行时间超过了long_query_time设定的阈值的 insert、update 等 DML 语句。

1.2 慢 SQL 的危害

从业务的角度来看：慢 SQL 会导致产品用户体验差，会减低用户对产品的好感度。

从数据库的角度来看：慢 SQL 会影响数据库的性能，每个 SQL 执行都需要消耗一定的 I/O 资源。假设总资源是100，有一条慢 SQL 占用了30的资源共计1分钟。那么在这1分钟时间内，其他 SQL 能够分配的资源总量就是70，如此循环，当资源分配完的时候，所有新的 SQL 执行将会排队等待。

二、慢SQL是怎么产生的？

【缺乏索引】：如果在查询中涉及到的列没有适当的索引，数据库系统可能需要执行全表扫描来找到匹配的行，从而导致查询变慢。

【查询条件不当】：查询条件过于复杂、使用了不必要的 JOIN 操作、存在子查询等，都可能导致查询性能下降。

【数据量过大】：当数据表中的数据量非常庞大时，即使有索引，查询也可能变得缓慢。

【锁等待】：如果查询需要访问被其他事务锁定的资源，就会导致查询阻塞，执行时间变长。

【硬件资源不足】：数据库服务器的硬件资源（如 CPU、内存、磁盘）不足以支撑查询的执行，也会导致查询变慢。

【不合适的数据库设计】：数据库表的设计不合理，如过度范式化、冗余数据等，会导致查询性能下降。

【统计信息不准确】：数据库的统计信息不准确会导致查询优化器做出错误的执行计划，影响查询性能。

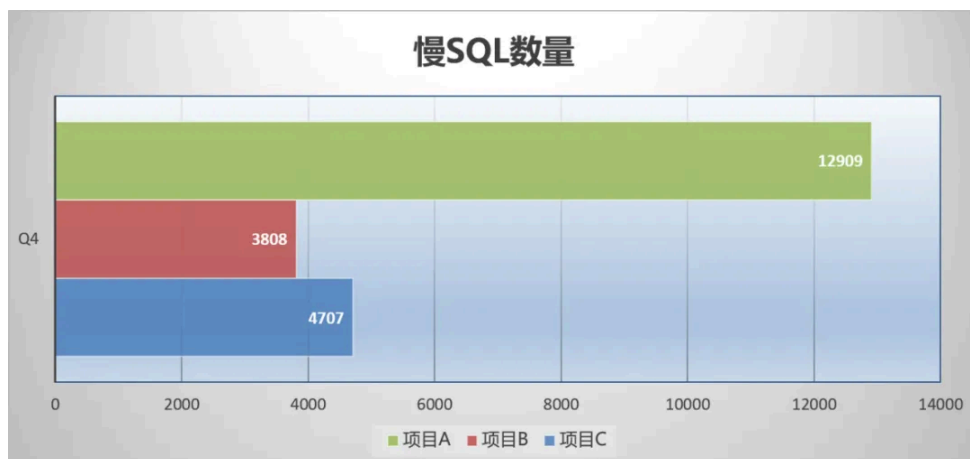
彩蛋提醒

我们为大家准备了抽奖福利，请继续阅读下去。

三、慢 SQL 治理实践

3.1 问题分析和解决方案

通过监控平台发现，项目的慢 SQL 数量较多，每天可能会产生几千甚至上万的慢 SQL，这对系统性能和用户体验都有不小的影响，因此，优化慢 SQL 问题值得被重视。我们从以下几个方面进行排查和分析：



1.数据量

我们都知道，同样的 SQL 语句，对于不同数据量的库表，查询效率也不一样，当数据量达到千万级甚至上亿，普通的查询语句执行时间可能也会超过一秒，进而出现慢 SQL 问题。

经过排查，发现不少库表的数据量已经达到千万，个别分表的数据量已经达到一亿多，几年前的历史数据仍然保留，需要进行人工清理。

针对数据量对 SQL 执行带来的影响，我们可以从三个方面解决：

(1) 清理数据

最直接的方式就是将无效数据清理，很多几年前的数据，几乎没有存储的价值，可以直接提交删除语句进行清理掉，当然，我们在删除时要注意线上影响，避免一次性删除太多数据，容易造成线上

数据库效率受到影响，对于单个分表可以采用分批删除，每一批只删除一个时间段；对于多个分表可以按照分表去删除，尽量减小对线上环境的影响。

以活动中台系统的答题活动为例，随着答题活动几年的运行，数据库已经有大量的用户数据，目前线上数据量很多到了千万级，个别到了上亿的数据量，这对线上访问影响很大。

所以我们进行手动清理数据，目前线上一共10张分表，考虑到对线上用户业务的影响，我们通过两方面减小影响：

- 一是分五次进行清理，每次只清理2~3张分表；
- 二是将删除语句转化为根据主键删除，大大提升 SQL 语句执行的效率。

最终清理掉大量的活动数据，我们采用的删除策略是将一年以前的历史数据进行删除，这部分数据已经不会用到，这个时间范围清理掉了大量的无效数据。

(2) 分库分表

手动清理数据虽然能解决问题，但是治标不治本，而且对于一些特殊的活动，可能留下的数据量仍然很大，这个时候如果库表的数据量仍然较大，且不能进行删除，就要考虑分库分表是否足够，可能之前出于业务考虑没有进行分表或者分表数量较少，这个时候就要考虑进行更多的分表，以提升数据库访问的效率。这里如果可以的话最好设置分表策略，建立配置项，灵活扩充分表，避免直接发版。

活动中台系统会有一张单独的路由表，记录分表路由，便于快速查询分表。

同时可以把路由相关配置直接建立在配置中心，无须发版就可以快速扩分表。可以根据数据量大小创建10、20、50、100张分表。

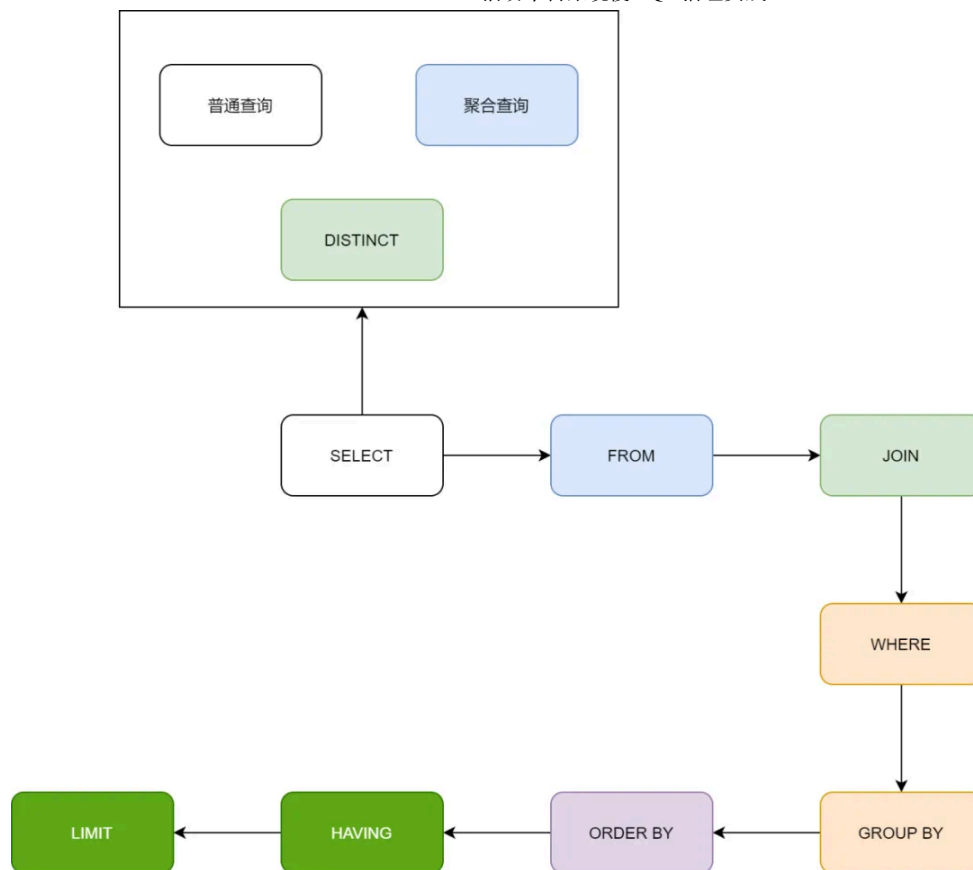
(3) 大数据量查询移步ES数据库

当然，对于一些响应要求比较高的业务需求，MySQL 数据库的性能可能无法达到要求，这个时候可以考虑将数据存储在 Elasticsearch 数据库或者缓存当中。

2.SQL 语句

数据库的数据量是重要的影响因素，但是 SQL 语句本身更会影响执行的效率，规范的 SQL 语句是避免慢 SQL 的前提。根据下图可以看出，SQL 的执行顺序为：

- 首先执行 from、join 来确定表之间的连接关系，得到初步的数据。
- 然后利用 where 关键字后面的条件对符合条件的语句进行筛选。
- from & join&where：用于确定要查询的表的范围，涉及到哪些表。



那么根据这个执行顺序，我们来看下 SQL 执行过程中常见的问题有哪些？

(1) 查询字段

我们在项目中查询最常用的是返回整个 DO 层对象，但其实很多时候我们只需要其中几个字段甚至一个字段，这个时候查询整个对象是很不划算的，比如下面这个例子，该语句在数据库管理器中的执行结果如下，执行时间为9269ms。

```
1 select * from a where id = 0;
```

然而我们只是想查询指定活动下的单个字段，这个时候可以不需要返回其他字段，只返回需要的字段，优化后示例如下，执行时间为4104ms，明显执行时间变短了。

```
1 select result from a where id = 0;
```

(2) 索引问题

索引在数据库查询中是一个很重要的影响因素，走不到索引和走到索引是有很大的区别的。但是索引有弊也有利：

优点

- 提高查询语句的执行效率，减少 IO 操作的次数
- 创建唯一性索引，可以保证数据库表中每一行数据的唯一性
- 加了索引的列会进行排序，在使用分组和排序子句进行查询时，可以显著减少查询中分组和排序的时间

缺点

- 索引需要占物理空间
- 创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加
- 当对表中的数据进行增删改查时，索引也要动态的维护，这样就降低了数据的更新效率
- 所以合理的设置索引并利用索引是高效执行SQL的重要因素。

以下这个案例来分析：

```

1 CREATE TABLE `table_test` (
2   `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
3   `test` varchar(128) NOT NULL DEFAULT '' COMMENT '索引测试字段',
4   .....
5   `result` varchar(128) NOT NULL DEFAULT '' COMMENT '结果测试字段',
6   PRIMARY KEY (`id`),
7   KEY `test` (`test`),
8 ) COMMENT='记录表';

```

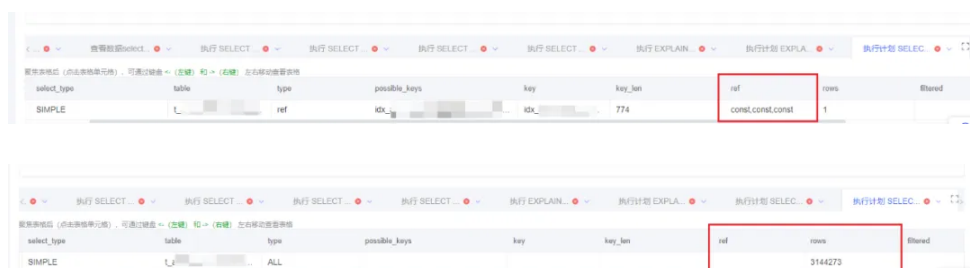
这是活动平台系统纪录的数据表，其中【result】是唯一的，这里可以看到索引没有涉及【result】，如果我们根据这个字段去查询唯一记录，看下执行结果如何：



耗时4286ms，这里根据建表语句可以看到有索引【KEY test (test)】，如果我们根据索引字段来查询该条记录的话，看下执行结果如何：



可以看到只需要16ms，索引对数据库效率的提升至关重要。我们可以根据 explain 关键字来解析一下 SQL 语句，比如刚才的案例，我们来看下相应的结果：



可以看到，直接根据非索引字段【result】查询，type 字段值为【ALL】，代表遍历全表，扫描行数为3144273，如果带上索引字段【test】，type 字段值为【ref】，代表匹配到单行记录值，扫描行数为1，显然通过索引查询效率得到了极大的提升。对于索引的使用会出现以下几种常见类型：

ALL：全表扫描，表示 MySQL 将遍历整个表以满足查询条件。这通常是效率最低的访问类型，应尽量避免。

index：索引全扫描，表示 MySQL 将遍历整个索引以满足查询条件，而不是遍历整个表。虽然比全表扫描效率要高，但仍然需要遍历索引的每一行。

range：范围扫描，表示 MySQL 使用了索引的一部分来满足查询条件，例如使用了索引的某个范围。这通常发生在有范围查询条件时，例如使用了 WHERE 子句中的 BETWEEN、>、< 等操作符。

index_merge：索引合并，表示 MySQL 使用了多个索引来满足查询条件，然后将结果合并。这通常发生在查询中有多个条件，每个条件可以使用不同的索引来访问数据。

unique_subquery：唯一子查询，表示 MySQL 使用了子查询来获取唯一的结果，并且子查询使用了唯一索引。

const：常量，表示 MySQL 使用了常量表来获取结果，这通常发生在查询条件中包含了常量值。

ref：引用，表示 MySQL 使用了非唯一索引来扫描表，通常发生在查询中使用了单个索引列作为条件。

（3）联表查询

【JOIN】关键词在项目的日常查询中可能会遇到，对于一些管理平台项目可能要求较低，可以作为日常开发查询语句，但是对于面向用户的项目，由于数据量较大，往往需要避免联合查询的使用。以下面这个案例来分析：

```
1 select * from a left join b on a.id = b.id where a.id = 0;
```

这个SQL是联合多张表进行联合查询，还有其他查询条件，执行得出结果为1432ms，超过1秒则为慢 SQL，这里联合两张表，如果相关的数据量较大，则执行速度会较慢。我们可以将 SQL 拆分为两个语句执行，拆为以下两个 SQL 分步执行：

```
1 select * from a where id = 0;
2 select * from b where id = 0;
```

分步获取数据库结果后再进行聚合，分步执行结果分别为728ms和744ms，联合查询拆分为简单查询可以有效减少慢 SQL，同时可以提高SQL查询的复用性。

（4）条件查询

在项目的日常开发中，SQL 语句切忌使用复杂查询，这会对数据库造成较大的压力。下面这个例子就是使用了比较复杂的条件查询：

```
1 select * from a where id in (select id from b) and time > '2024-03-29';
```

使用查询条件的同时还嵌套了查询语句，除此之外还夹杂了联合查询，语句已经比较复杂，我们来看下查询结果：



可以看到执行时间非常缓慢，达到12648ms，这个SQL的背景是用于业务对账，虽然使用的是离线数据库，但是仍然不能忽视它的风险，每天大量的执行这种复杂 SQL，还是对业务有一定的风险影响。考虑到业务价值单一，复用性不高，我们可以直接冗余一份数据到单独的表里，避免复杂查询，直接一步到位解决对账带来的慢 SQL 问题。

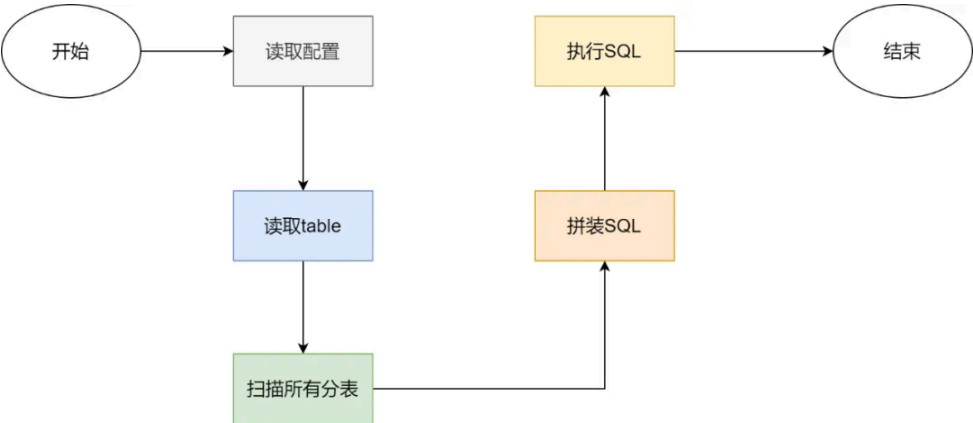
3. 整体策略

清理完数据库的无效数据，优化完 SQL 语句本身，可能还是会出现慢 SQL 问题，这个时候我们要考虑下，是否可以优化整体的数据库交互策略，以活动中台系统的数据清理慢 SQL 为例。

活动中台系统创建了一个定时任务，每天凌晨执行一次，对数据库的无效历史数据进行统一清理，具体删除哪些库表、什么时间段、什么条件都由配置项灵活控制，配置项示例如下：

```
1 "分表数量": 7,  
2 "表名": table,  
3 "条件": condition
```

包括分表的数量、要删除数据的表名、查询的条件信息，查询条件对应SQL语句中的【where】信息。执行流程如下图：



整体上看删除策略很通用，条件配置灵活，可以同时应对不同分表、不同查询条件等。但是线上运行发现会产生大量的慢SQL，可能每天就会产生几千条。

主要原因如下：

- 会出现联表查询的情况，数据量较大的表会出现执行时间超过一秒的 SQL 语句；
- 删除策略中时间是重要的因素，但是时间往往不会设置为索引字段，所以很难充分利用索引；
- 删除策略会扫描所有分表，但是很多分表可能全部扫描也没有需要删除的数据，会出现无效执行的情况。

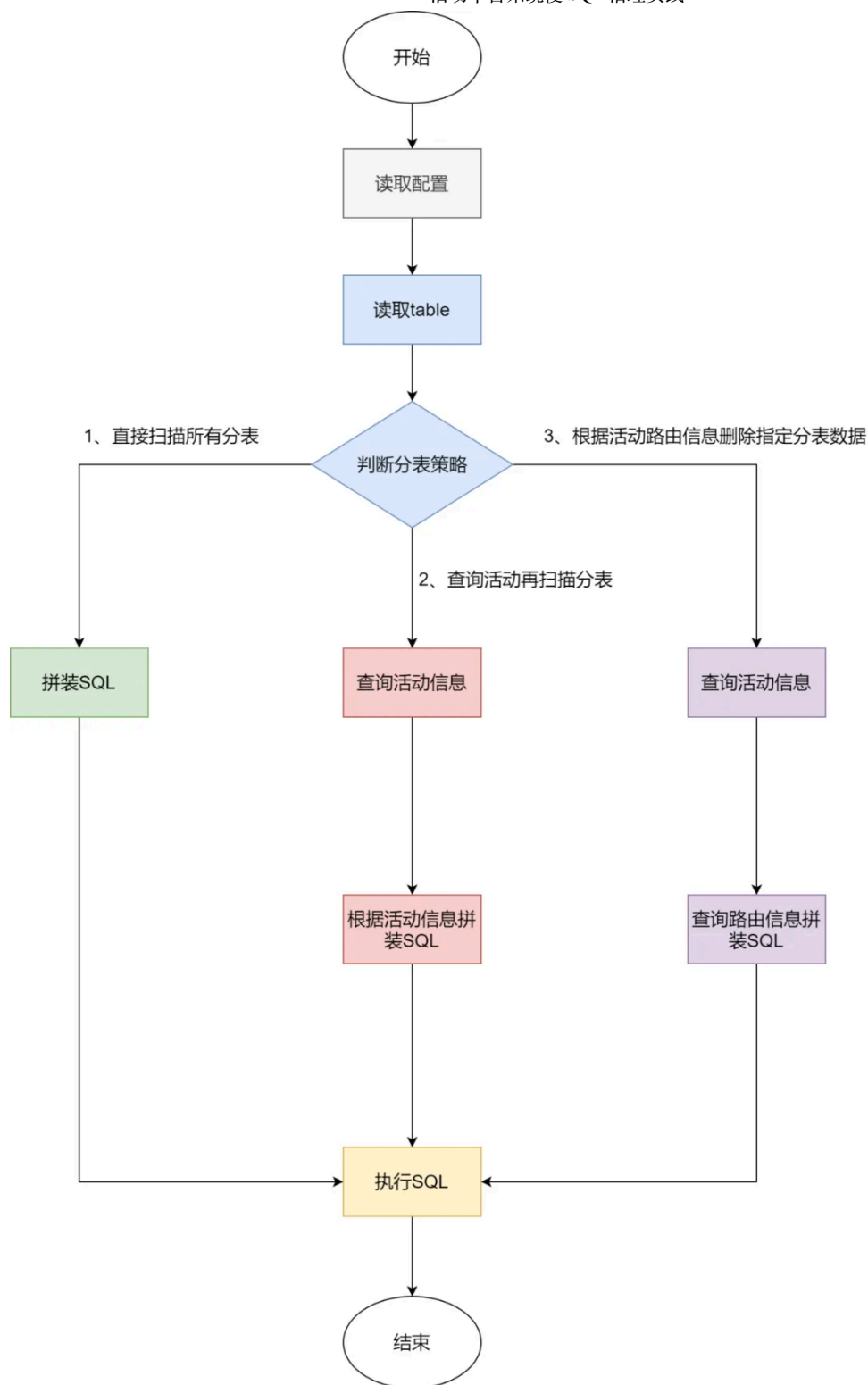
针对以上策略，我们进行了综合改进，主要措施有以下几条：

- 避免联表查询，尽可能的拆分为简单 SQL 执行；
- 不要从时间维度直接出发，而是从活动维度出发，如果一个活动结束时间较长，那么这个活动相关的数据自然不再需要，可以直接删除；
- 利用分表路由信息，从活动本身出发快速路由到相关分表，减少 SQL 语句的无效执行。

修改完的配置项如下：

```
1  "分表数量": 7,  
2  "表名": table,  
3  "条件": condition,  
4  "删除数量": 1,  
5  "删除策略": 1,  
6  "开始时间": "",  
7  "结束时间": ""
```

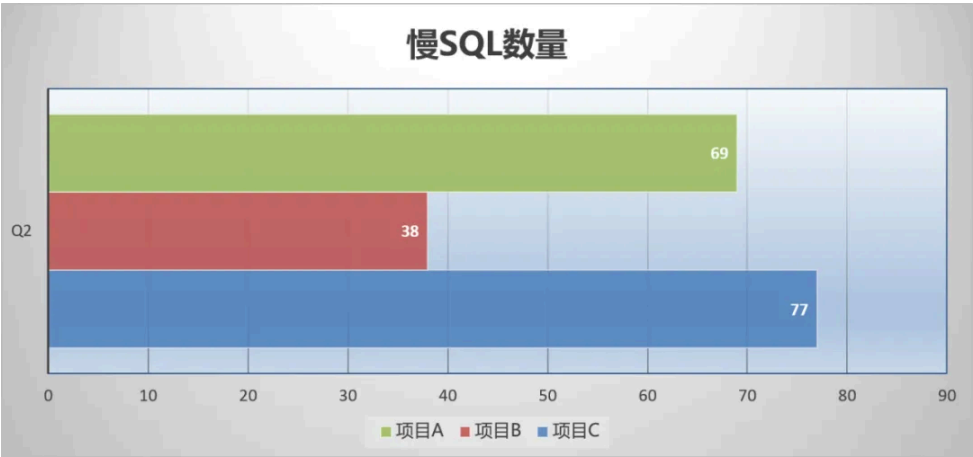
包括分表的数量、要删除数据的表名、查询条件、每次删除的数量、开始时间和结束时间，在这个时间范围内的活动都符合条件，整体上看删除策略更为通用，条件配置更加灵活。优化的删除策略流程图如下：



从图中可以看到，我们保留原有的删除策略，避免个别数据表没有活动信息和分表信息，只能单纯根据时间扫描删除；除此之外，对于有活动信息的数据，我们会先查询活动，根据活动去删除，如果数据表是根据活动进行分表，则直接查询路由信息，并删除指定分表的数据即可，也就是图中的第三种策略，线上大部分的数据都会按照第三种策略执行，如果有活动信息但是未按照活动进行分表，可能按照用户等维度进行分表，则可以根据活动信息进行删除，也可以有效避免慢 SQL 的出现。

3.2 治理效果

经过对慢 SQL 的专项治理，活动中台系统的慢 SQL 数量由几千个，稳定在了两位数，有效减少了慢 SQL 的数量，进一步提升了系统稳定性。



四、经验总结

- 治理慢 SQL 的根本是从源头避免慢 SQL，项目组内部必须达成高度一致，根据编码规范进行前置避免慢 SQL 的出现。
- 离线数据库不能成为忽视慢 SQL 问题的原因，仍然会有影响线上业务的风险，数据库实例如果是混合部署的方式，可能离线库所在的机器有其他业务的主库，而且如果从库延迟严重会影响主从故障切换。
- 数据库的合理设计不能依赖后期重构，一开始就要尽可能的考虑充分。
- 慢 SQL 治理过程中出现的问题可以及时复盘，避免团队其他成员继续踩坑。

..... END

vivo
互联网技术

粉丝福利

vivo互联网技术为开发者朋友们提供了抽奖福利

参与方式：

1. 转发本文到[朋友圈](#)，截图发送到“vivo互联网技术”公众号后台；
2. 在公众号后台回复“[v爱技术](#)”即可获取抽奖资格。

开奖时间：2025年4月25日 20:00

奖品：



iQOO & NBA 联名帆布袋 x 2个

注意：


1. 中奖者需在开奖5天内提供收货信息（联系本号后台）。
2. 我们将会通过人工确认为符合中奖条件者邮寄奖品。
3. 未及时提供收货信息或未完成所有步骤的参与者将视为自动放弃。

活动解释权归vivo互联网技术所有




2025年3月中奖名单

iQOO & NBA 联名洗漱包, 1 人中奖



光烛

iQOO & NBA 联名帆布袋, 1 人中奖



Zhiyahan

猜你喜欢

- 深度剖析 StarRocks 读取 ORC 加密文件背后的技术
- 缓存监控治理在游戏业务的实践和探索
- vivo 大规模容器集群运维平台实践



vivo互联网技术

分享 vivo 互联网技术干货与沙龙活动，推荐最新行业动态与热门会议。
464篇原创内容

服务号

服务器 · 目录

上一篇

缓存监控治理在游戏业务的实践和探索

下一篇

Full GC 频率优化实战

修改于2025年04月17日

