

## 14 | 订单数据越来越多，数据库越来越慢该怎么办？

李玥 · 后端存储实战课



你好，我是李玥。

在前面几节课，我们一起学习了在并发持续高速增长的情况下，如何来逐步升级存储。今天这节课我们来聊一聊，如何应对数据的持续增长，特别是像订单数据这种会随着时间一直累积的数据。

为什么数据量越大数据库就越慢？你得理解这里面的根本原因。

我们知道，无论是“增删改查”哪个操作，其实都是查找问题，因为你都得先找到数据才能对数据做操作。那存储系统性能问题，其实就是查找快慢的问题。

无论是什么样的存储系统，一次查询所耗费的时间，都取决于两个因素：

1. 查找的时间复杂度；
2. 数据总量。

这也是为什么大厂面试时总喜欢问“时间复杂度”相关问题的原因。查找的时间复杂度又取决于两个因素：

1. 查找算法；
2. 存储数据的数据结构。

你看，这两个知识点也是面试问题中的常客吧？所以人家面试官并不是非要问你一些用不上的问题来为难你，这些知识点真的不是用不上，而是你不知道怎么用。

我们把话题拉回来。对于我们大多数做业务的系统，用的都是现成的数据库，数据的存储结构和查找算法都是由数据库来实现的，业务系统基本没法去改变它。比如说，我们讲过 MySQL 的 InnoDB 存储引擎，它的存储结构是 B+ 树，查找算法大多就是树的查找，查找的时间复杂度就是  $O(\log n)$ ，这些都是固定的。那我们唯一能改变的，就是数据总量了。

所以，**解决海量数据导致存储系统慢的问题，思想非常简单，就是一个“拆”字，把一大坨数据拆分成 N 个小坨，学名叫“分片（Shard）”**。拆开之后，每个分片里的数据就没那么多了，然后让查找尽量落在某一个分片上，这样来提升查找性能。

所有分布式存储系统解决海量数据查找问题都是遵循的这个思想，但是光有思想还不够，还需要落地，下面我们就来说如何拆分数据的问题。

## 存档历史订单数据提升查询性能

我们在开发业务系统的时候，很多数据都是具备时间属性的，并且随着系统运行，累计增长越来越多，数据量达到一定程度就会越来越慢，比如说电商中的订单数据，就是这种情况。按照我们刚刚说的思想，这个时候就需要拆分数据了。

我们的订单数据一般都是保存在 MySQL 中的订单表里面，说到拆分 MySQL 的表，大多数同学的第一反应都是“分库分表”，别着急，咱现在的数据量还没到非得分库分表那一步呢，下一节课我会和你讲分库分表。**当单表的订单数据太多，多到影响性能的时候，首选的方案是，归档历史订单。**

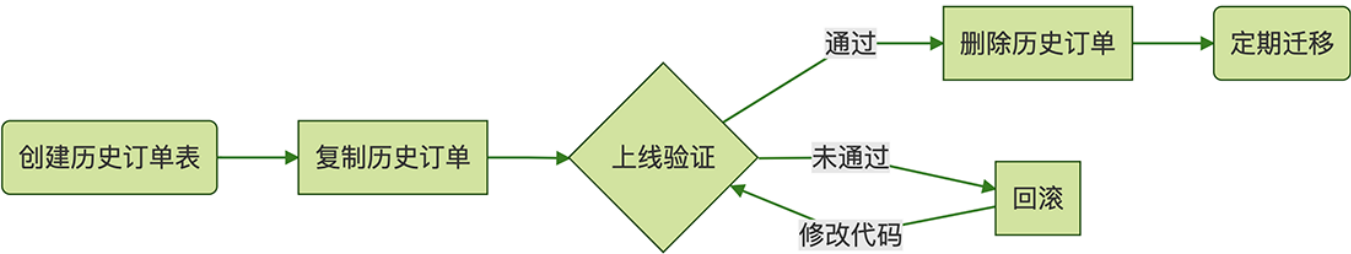
所谓归档，其实也是一种拆分数据的策略。简单地说，就是把大量的历史订单移到另外一张历史订单表中。为什么这么做呢？因为像订单这类具有时间属性的数据，都存在热尾效应。大多数情况下访问的都是最近的数据，但订单表里面大量的数据都是不怎么常用的老数据。

因为新数据只占数据总量中很少的一部分，所以把新老数据分开之后，新数据的数据量就会少很多，查询速度也就会快很多。老数据虽然和之前比起来没少多少，查询速度提升不明显，但是，因为老数据很少会被访问到，所以慢一点儿也问题不大。

这样拆分的另外一个好处是，**拆分订单时，需要改动的代码非常少**。大部分对订单表的操作都是在订单完成之前，这些业务逻辑都是完全不用修改的。即使像退货退款这类订单完成后的操作，也是有时限的，那这些业务逻辑也不需要修改，原来该怎么操作订单表还怎么操作。

基本上只有查询统计类的功能，会查到历史订单，这些需要稍微做一些调整，按照时间，选择去订单表还是历史订单表查询就可以了。很多电商大厂在它逐步发展壮大的过程中，都用这种订单拆分的方案撑了好多年。你可能还有印象，几年前你在京东、淘宝查自己的订单时，都有一个查“三个月前订单”的选项，其实就是查订单历史表。

归档历史订单，大致的流程是这样的：



1. 首先我们需要创建一个和订单表结构一模一样的历史订单表；
2. 然后，把订单表中的历史订单数据分批查出来，插入到历史订单表中去。这个过程你怎么实现都可以，用存储过程、写个脚本或者写个导数据的小程序都行，用你最熟悉的方法就行。如果你的数据库已经做了主从分离，那最好是去从库查询订单，再写到主库的历史订单表中去，这样对主库的压力会小一点儿。


3. 现在，订单表和历史订单表都有历史订单数据，先不要着急去删除订单表中的数据，你应该测试和上线支持历史订单表的新版本代码。因为两个表都有历史订单，所以现在这个数据库可以支持新旧两个版本的代码，如果新版本的代码有 Bug，你还可以立刻回滚到旧版本，不至于影响线上业务。
4. 等新版本代码上线并验证无误之后，就可以删除订单表中的历史订单数据了。
5. 最后，还需要上线一个迁移数据的程序或者脚本，定期把过期的订单从订单表搬到历史订单表中去。

类似于订单商品表这类订单的相关的子表，也是需要按照同样的方式归档到各自的历史表中，由于它们都是用订单 ID 作为外键来关联到订单主表的，随着订单主表中的订单一起归档就可以了。

这个过程中，我们要注意的问题是，要做到对线上业务的影响尽量的小。迁移这么大量的数据，或多或少都会影响数据库的性能，你应该尽量放在闲时去迁移，**迁移之前一定做好备份**，这样如果不小心误操作了，也能用备份来恢复。


## 如何批量删除大量数据？

这里面还有一个很重要的细节问题：如何从订单表中删除已经迁走的历史订单数据？我们直接执行一个删除历史订单的 SQL 行不行？像这样删除三个月前的订单：

 复制代码

```
1 delete from orders
2 where timestamp < SUBDATE(CURDATE(),INTERVAL 3 month);
```

大概率你会遇到错误，提示删除失败，因为需要删除的数据量太大了，所以需要分批删除。比如说我们每批删除 1000 条记录，那分批删除的 SQL 可以这样写：


 复制代码

```
1 delete from orders
2 where timestamp < SUBDATE(CURDATE(),INTERVAL 3 month)
3 order by id limit 1000;
```

执行删除语句的时候，最好在每次删除之间停顿一会儿，避免给数据库造成太大的压力。上面这个删除语句已经可以用了，反复执行这个 SQL，直到全部历史订单都被删除，是可以完成删除任务的。

但是这个 SQL 还有优化空间，它每执行一次，都要先去 timestamp 对应的索引上找出符合条件的记录，然后再把这些记录按照订单 ID 排序，之后删除前 1000 条记录。

其实没有必要每次都按照 timestamp 比较订单，所以我们可以先通过一次查询，找到符合条件的历史订单中最大的那个订单 ID，然后在删除语句中把删除的条件转换成按主键删除。

 复制代码

```
1 select max(id) from orders
2 where timestamp < SUBDATE(CURDATE(),INTERVAL 3 month);
3
4
5 delete from orders
6 where id <= ?
7 order by id limit 1000;
```

这样每次删除的时候，由于条件变成了主键比较，我们知道在 MySQL 的 InnoDB 存储引擎中，表数据结构就是按照主键组织的一颗 B+ 树，而 B+ 树本身就是有序的，所以不仅查找非常快，也不需要再进行额外的排序操作了。当然这样做的前提条件是订单 ID 必须和订单时间正相关才行，大多数订单 ID 的生成规则都可以满足这个条件，所以问题不大。

然后我们再说一下，为什么在删除语句中非得加一个排序呢？因为按 ID 排序后，我们每批删除的记录，基本都是 ID 连续的一批记录，由于 B+ 树的有序性，这些 ID 相近的记录，在磁盘的物理文件上，大致也是放在一起的，这样删除效率会比较高，也便于 MySQL 回收页。

大量的历史订单数据删除完成之后，如果你检查一下 MySQL 占用的磁盘空间，你会发现它占用的磁盘空间并没有变小，这是什么原因呢？这也是和 InnoDB 的物理存储结构有关系。

虽然逻辑上每个表是一颗 B+ 树，但是物理上，每条记录都是存放在磁盘文件中的，这些记录通过一些位置指针来组织成一颗 B+ 树。当 MySQL 删除一条记录的时候，只能是找到记录所



在的文件中位置，然后把文件的这块区域标记为空闲，然后再修改 B+ 树中相关的一些指针，完成删除。其实那条被删除的记录还是躺在那个文件的那个位置，所以并不会释放磁盘空间。


这么做也是没有办法的办法，因为文件就是一段连续的二进制字节，类似于数组，它不支持从文件中间删除一部分数据。如果非要这么删除，只能是把这个位置之后的所有数据往前挪，这样等于是移动大量数据，非常非常慢。所以，删除的时候，只能是标记一下，并不真正删除，后续写入新数据的时候再重用这块儿空间。

理解了原理，你就很容易知道，不仅是 MySQL，很多其他的数据库都会有类似的问题。这个问题也没什么特别好的办法解决，磁盘空间足够的话，就这样吧，至少数据删了，查询速度也快了，基本上是达到了目的。

如果说我们数据库的磁盘空间很紧张，非要把这部分磁盘空间释放出来，可以执行一次 OPTIMIZE TABLE 释放存储空间。对于 InnoDB 来说，执行 OPTIMIZE TABLE 实际上就是把表重建一遍，执行过程中会一直锁表，也就是说这个时候下单都会被卡住，这个是需要注意的。另外，这么优化有个前提条件，MySQL 的配置必须是每个表独立一个表空间（innodb\_file\_per\_table = ON），如果所有表都是放在一起的，执行 OPTIMIZE TABLE 也不会释放磁盘空间。

重建表的过程中，索引也会重建，这样表数据和索引数据都会更紧凑，不仅占用磁盘空间更小，查询效率也会有提升。那对于频繁插入删除大量数据的这种表，如果能接受锁表，定期执行 OPTIMIZE TABLE 是非常有必要的。

如果说，我们的系统可以接受暂时停服，最快的方法是这样的：直接新建一个临时订单表，然后把当前订单复制到临时订单表中，再把旧的订单表改名，最后把临时订单表的表名改成正式订单表。这样，相当于我们手工把订单表重建了一次，但是，不需要漫长的删除历史订单的过程了。我把执行过程的 SQL 放在下面供你参考：

 复制代码

```
1 -- 新建一个临时订单表
2 create table orders_temp like orders;
3
4
5 -- 把当前订单复制到临时订单表中
```

```
6 insert into orders_temp
7   select * from orders
8   where timestamp >= SUBDATE(CURDATE(),INTERVAL 3 month);
9
10
11 -- 修改替换表名
12 rename table orders to orders_to_be_dropd, orders_temp to orders;
13
14
15 -- 删除旧表
16 drop table orders_to_be_dropd
```

## 小结

对于订单这类具有时间属性的数据，会随时间累积，数据量越来越多，为了提升查询性能需要对数据进行拆分，首选的拆分方法是把旧数据归档到历史表中去。这种拆分方法能起到很好的效果，更重要的是对系统的改动小，升级成本低。

在迁移历史数据过程中，如果可以停服，最快的方式是重建一张新的订单表，然后把三个月内的订单数据复制到新订单表中，再通过修改表名让新的订单表生效。如果只能在线迁移，那需要分批迭代删除历史订单数据，删除的时候注意控制删除节奏，避免给线上数据库造成太大压力。

最后，我要再一次提醒你，线上数据操作非常危险，在操作之前一定要做好数据备份。

## 思考题

在数据持续增长的过程中，今天介绍的这种“归档历史订单”的数据拆分方法，和直接进行分库分表相比，比如说按照订单创建时间，自动拆分成每个月一张表，两种方法各有什么优点和缺点？欢迎你在留言区与我讨论。

感谢你的阅读，如果你觉得今天的内容对你有帮助，也欢迎把它分享给你的朋友。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

## 精选留言 (40)



李玥 置顶

2020-03-30

Hi, 我是李玥。

这里回顾一下上节课的思考题：

课后请你想一下，复制状态机除了用于数据库的备份和复制以外，在计算机技术领域，还有哪些地方也用到了复制状态机？欢迎你在留言区与我讨论。感谢你的阅读，如果你觉得今天的内容对你有帮助，也欢迎把它分享给你的朋友。

复制状态机的应用是非常广泛的，比如说现在很火的区块链技术，也是借鉴了复制状态机理论，它的链，或者说是账本就是操作日志，每个人的钱包，就是状态。它只要保证账本一旦记录后就不会被篡改，那在任何人的电脑上，计算出来的钱包就都是一样的。



👍 30



约书亚

2020-03-28

如果不进行OPTIMIZE，想通过历史表来提升性能的目的岂不是达不到了？

作者回复：不执行OPTIMIZE也是可以提升性能的。数据和索引虽然在物理上没有删除，但逻辑上已经删除掉了，执行查询操作的时候，并不会去访问这些已经删除的数据。

比如，原来有100条数据，删除完成后剩了10条。虽然100条数据都在磁盘文件中，但这时候执行一次全表扫描，MySQL只会访问剩下的10条数据。

共 3 条评论 >

👍 33



AI

2020-04-27

一下子和Java的GC算法产生了共鸣。

创建新表的方式，只复制少部分数据，效率更高，但你要能接受这段时间的STW。这是复制算法。

历史归档，删除数据的方式，会产生碎片，利用率低。只有到空间不足的情况下，才进行压缩



整理 (OPTIMIZE) 。这是标记清理算法，关键时刻再整理 (STW) ， CMS GC就是这个思路。

作者回复: 是的，磁盘碎片和内存碎片产生的原因是一样的，所以清理的思路很多都是相似的。

共 3 条评论 >

👍 31



webmin

2020-03-28

“归档历史订单”可以灵活控制，比如把不再会进行修改的订单，迁移到偏重查询快系统（各种NOSQL），不再需要online查询的数据，可以迁移到offline的库中。

直接进行分库分表，会遇到冷热不均的问题，如：电商大促或年节购物旺季订单量与淡季和平季订单可能会量级差别。用时间这个维度去分库分表，操作上相对简单，但是到达这种需要分库分表量级的系统，切分的灵活性更加重要，怎么分业务场景不同切分维度也会不同。

共 2 条评论 >

👍 30



攻城拔寨

2020-03-28

1. 自动分表需要事先做好预估，把时间间隔设置好，如果表数据增长速度不均匀（例如淡季旺季，后期业务膨胀），可能需要重新设计分表规则，很麻烦。表名也变化了，代码侵入性比较大。

优点就是如果数据增长速度变化不大，不用持续做归档。

2. 归档的好处是代码侵入性低，因为表名字还是一样。表增长速度变成也能灵活改变归档数据大小跟速度。

缺点就是需要持续归档迁移，后期归档数据太大也会遇到瓶颈

共 1 条评论 >

👍 13



刘楠

2020-03-28

alter table A engine=InnoDB 命令来重建这样也能达到释放空间的效果吧

作者回复: 是可以的。

共 6 条评论 >

👍 11



乖，摸摸头

2020-05-12

按时间分库分表一直有个疑惑，按月进行分表，有几个月数据很小，有几个月数据特别大，这种会怎么处理

作者回复：这种情况可能就不适合按月来分片。



8



leslie

2020-03-29

定期归历史的方式其实oracle的使用频率是最高的。

不过删主键的方式确实不曾想到和尝试过，觉得短期不失为不错的选择。

自动分表总归会有坑：这也是为何自动化的极限还是要人去监控；自动化减少的人的机械操作而已，不是不需要人去操作和监控，尤其数据库数量众多时还是要人去自动化。

手工虽麻烦细节上的把控会比较好：细节会把控的比较好。

相对合理的方式应当是二者结合：1) 拆分之前人为的做一次检查，2) 拆分的动作自动化去执行，3) 结果由人去复核。

毕竟当需要同时拆分的工作量很庞大时不可能全部都是手工操作，这其实就像运维：一个人去操作2-3台可能，20-50其实就很困难了，500以上要全部人为操作基本就只能自动化+人为操作了。

谢谢今天的分享：期待后续的课程。



7



此方彼方Francis

2020-03-28

任何时间属性相关的数据基本都可以这样处理（比如聊天数据），这种处理办法和分库分表等并不冲突。这种办法相对简单大部分情况下效果也比较好，只是如果业务发展很好，那么订单表的数据依旧有可能很多，另外历史数据表依旧是需要查询的，时间越久数据量越多，查询历史数据太慢的话迟早也会是个问题。

分库分表这种方案需要选比较好的shard key，在数据统计上会麻烦一些，单表数据量上来之后依旧有归档的需求。

按月新建表会有数据热点问题，查询和统计还是会比较麻烦。



7



王

2020-03-28

老师，什么情况下需要归档，什么情况需要分库分表呢？有什么具体的指标吗？

作者回复: 我个人的建议是, 如果归档能解决问题, 就不要分库分表。我们下节课会详细讲讲分库分表。

共 2 条评论 >

👍 5



**Monday**

2020-08-02

你可能还有印象, 几年前你在京东、淘宝查自己的订单时, 都有一个查“三个月前订单”的选项, 其实就是查订单历史表。

这个确实有印象, 以前我们产品也抄了这个查询条件。不过我的问题是, 现在怎么没有这个选项了?

共 4 条评论 >

👍 4



**怀朔**

2020-03-28

比这个删除规律 是不是有问题啊?

- 1、创建temp表
- 2、把历史数据迁移老表
- 3、check历史数据条数
- 4、删除老数据?

按照老师的方法

在rename 之前 插入之后 这时候有数据进来 数据会丢在老表里面了?

作者回复: 所以我们说, 这个方案的前提是必须得停机操作。

共 3 条评论 >

👍 4



**Jxin**

2020-04-04

1.前者操作所有数据都会有一个路由的前置操作, 这是有开销的。其次, 因为分表了, 所以其区域查找这种就需要多表数据做聚合, 这就会让查询变得很复杂低效(采用mycat这种中间件可以规避业务代码大量改动的问题, 但聚合数据的开销依旧是跑不掉的, 而且引入中间件还有多一条的问题)。

2.后者其实性能, 操作成本, 对业务代码的侵入程度都比前者有优势, 唯一遗憾的就是其应用

场景有限。只有在整体业务单量不大，且归档数据操作概率极低的情况下适用。因为如果业务单量很大，比如日单量一百万，那么这个热数据表能存多久的数据？十天？二十天？这样的时间区间是严重不符合归档条件的，往往归档数据都是半年，少说三四个月的完结订单。而且归档数据，数据量很大。意味着性能很差，频繁导入导出，查询修改，是支撑不住的。



Mq

2020-03-29

归档历史数据的优点是简单，对系统的改造少，缺点是不是长久之计  
分库分表需要对数据访问层做架构变更，对系统的改造大，要考虑数据分布，对接口查询性能等业务需求的影响，另外我觉得按时间分表跟我们设计这个分库分表就不符，我们做业务数据分库分表就是想数据打散，按时间分达不到这个目的，按时间适用在做一体化系统，因为这些系统有很多报表统计需求可能用的上



夏目

2020-04-10

分库分表不需要迁移数据，容量也大，缺点是各个表数据可能差异较大，查询较麻烦。历史订单表查询相对简单，迁移数据麻烦



特种流氓

2020-03-29

最近的订单表往归档表挪数据的过程中可能一份数据在两张表都存在 这个时候用户查询全部订单的时候是否我们在应用利用是用去重去剔除重复数据

作者回复: 如果要同时查二个表，那合并和去重就在所难免。一般情况下，最好能设计好业务逻辑，尽量不要同时查当前和历史表。



每天晒白牙

2020-03-28

思考题

把历史订单数据归档方案实现相对简单，也很有效果，需要注意的就是归档时注意对线上服务的影响。

如果采用按照订单创建时间分库分表，优点是省去了后面归档历史数据的重复工作，在一定程度上可以提高写入和查询性能，但也有不足。

首先就是采用分库分表在技术复杂度上相比历史数据归档还是高一些的

其次就是如果刚好到新的一个月，前一个月的数据还是属于热数据，所以会涉及多表查询

最后就是这种方案会造成产生大量的表，如果订单数据不大，每个表中数据量也不会太大，有点浪费资源了



👍 2



**ifelse**

2022-12-10 来自浙江

解决海量数据导致存储系统慢的问题，思想非常简单，就是一个“拆”字，把一大坨数据拆分成 N 个小坨，学名叫“分片（Shard）”。--记下来



👍 2



**lesserror**

2020-05-24

老师，您的这句话：“其实每次都排序是没必要的，所以我们可以先通过一次查询，找到符合条件的历史订单中最大的那个订单 ID，然后在删除语句中把删除的条件转换成按主键删除。”

事实上每次，都做了排序啊？为什么说没必要呢？

作者回复: 这个地方改为：“其实没有必要每次都按照timestamp比较订单”。我联系编辑小姐姐修改，感谢你的问题！



👍 2



**Panmax**

2020-04-02

最后一种方案中（重建一张新的订单表），最后不应该把 orders\_to\_be\_droppd 表删掉吧，删掉之后历史数据就丢了。

作者回复: 这个可以灵活处理。



👍 1

