

15 | MySQL存储海量数据的最后一招：分库分表

2020-03-31 李玥

后端存储实战课

[进入课程 >](#)



讲述：李玥

时长 16:18 大小 11.21M



你好，我是李玥。

从这节课开始，我们课程将进入最后一部分“海量数据篇”，这节课也是我们最后一节主要讲 MySQL 的课程。解决海量数据的问题，必须要用到分布式的存储集群，因为 MySQL 本质上是一个单机数据库，所以很多场景下不是太适合存 TB 级别以上的数据。

但是，绝大部分的电商大厂，它的在线交易这部分的业务，比如说，订单、支付相关的系统，还是舍弃不了 MySQL，原因是，**只有 MySQL 这类关系型数据库，才能提供金融事务保证**。我们之前也讲过分布式事务，那些新的分布式数据库提供的所谓的分布式事务，多少都有点儿残血，目前还达不到这些交易类系统对数据一致性的要求。

那既然 MySQL 支持不了这么大的数据量，这么高的并发，还必须要用它，怎么解决这个问题呢？还是按照我上节课跟你说的思想，**分片**，也就是拆分数据。1TB 的数据，一个库撑不住，我把它拆成 100 个库，每个库就只有 10GB 的数据了，这不就可以了么？这种拆分就是所谓的 MySQL 分库分表。

不过，思路是这样没错，分库分表实践起来是非常不容易的，有很多问题需要去思考和解决。

如何规划分库分表？

还是拿咱们的“老熟人”订单表来举例子。首先需要思考的问题是，分库还是分表？分库呢，就是把数据拆分到不同的 MySQL 库中去，分表就是把数据拆分到同一个库的多张表里面。

在考虑到底是分库还是分表之前，我们需要先明确一个原则，**那就是能不拆就不拆，能少拆不多拆**。原因也很简单，你把数据拆分得越散，开发和维护起来就越麻烦，系统出问题的概率就越大。

基于这个原则我们想一下，什么情况下适合分表，什么情况下不得不分库？

那我们分库分表的目的是为了解决两个问题：

第一，是数据量太大查询慢的问题。这里面我们讲的“查询”其实主要是事务中的查询和更新操作，因为只读的查询可以通过缓存和主从分离来解决，这个我们在之前的“MySQL 如何应对高并发”的两节课中都讲过。那我们上节课也讲到过，**解决查询慢，只要减少每次查询的数据总量就可以了，也就是说，分表就可以解决问题**。

第二，是为了应对高并发的的问题。应对高并发的思想我们之前也说过，一个数据库实例撑不住，就把并发请求分散到多个实例中去，所以，**解决高并发的需要是分库的**。

简单地说，**数据量大，就分表；并发高，就分库**。

一般情况下，我们的方案都需要同时做分库分表，这时候分多少个库，多少张表，分别用预估的并发量和数据量来计算就可以了。

另外，我个人不建议你在方案中考虑二次扩容的问题，也就是考虑未来的数据量，把这次分库分表设计的容量都填满了之后，数据如何再次分裂的问题。

现在技术和业务变化这么快，等真正到了那个时候，业务早就变了，可能新的技术也出来了，你之前设计的二次扩容方案大概率是用不上的，所以没必要为了这个而增加方案的复杂程度。还是那句话，**越简单的设计可靠性越高**。

如何选择 Sharding Key?

分库分表还有一个重要的问题是，选择一个合适的列或者说是属性，作为分表的依据，这个属性一般称为 Sharding Key。像我们上节课讲到的归档历史订单的方法，它的 Sharding Key 就是订单完成时间。每次查询的时候，查询条件中必须带上这个时间，我们的程序就知道，三个月以前的数据查订单历史表，三个月内的数据查订单表，这就是一个简单的按照时间范围来分片的算法。

选择合适 Sharding Key 和分片算法非常重要，直接影响了分库分表的效果。我们首先来说如何选择 Sharding Key 的问题。

选择这个 Sharding Key 最重要的参考因素是，我们的业务是如何访问数据的。

比如我们把订单 ID 作为 Sharding Key 来拆分订单表，那拆分之后，如果我们按照订单 ID 来查订单，就需要先根据订单 ID 和分片算法计算出，我要查的这个订单它在哪个分片上，也就是哪个库哪张表中，然后再去那个分片执行查询就可以了。

但是，当我打开“我的订单”这个页面的时候，它的查询条件是用户 ID，这里没有订单 ID，那就没法知道我们要查的订单在哪个分片上，就没法查了。当然你要强行查的话，那就只能把所有分片都查一遍，再合并查询结果，这个就很麻烦，而且性能很差，还不能分页。

那要是把用户 ID 作为 Sharding Key 呢？也会面临同样的问题，使用订单 ID 作为查询条件来查订单的时候，就没办法找到订单在哪个分片了。这个问题的解决办法是，在生成订单 ID 的时候，把用户 ID 的后几位作为订单 ID 的一部分，比如说，可以规定，18 位订单号中，第 10-14 位是用户 ID 的后四位，这样按订单 ID 查询的时候，就可以根据订单 ID 中的用户 ID 找到分片。

那我们系统对订单的查询方式，肯定不只是按订单 ID 或者按用户 ID 这两种啊。比如说，商家希望看到的是自己店铺的订单，还有各种和订单相关的报表。对于这些查询需求，我们一旦对订单做了分库分表，就没法解决了。那怎么办呢？

一般的做法是，把订单数据同步到其他的存储系统中去，在其他的存储系统里面解决问题。比如说，我们可以再构建一个以店铺 ID 作为 Sharding Key 的只读订单库，专门供商家来使用。或者，把订单数据同步到 HDFS 中，然后用一些大数据技术来生成订单相关的报表。

所以你看，一旦做了分库分表，就会极大地限制数据库的查询能力，之前很简单的查询，分库分表之后，可能就没法实现了。所以我们在之前的课程中，先讲了各种各样的方法，来缓解数据多、并发高的问题，而一直没讲分库分表。**分库分表一定是，数据量和并发大到所有招数都不好使了，我们才拿出来的最后一招。**

如何选择分片算法？

在上节课我给你留的思考题中，我们提到过，能不能用订单完成时间作为 Sharding Key 呢？比如说，我分 12 个分片，每个月一个分片，这样对查询的兼容要好很多，毕竟查询条件中带上时间范围，让查询只落到某一个分片上，还是比较容易的，我在查询界面上强制用户必须指定时间范围就行了。

这种做法有个很大的问题，比如现在是 3 月份，那基本上所有的查询都集中在 3 月份这个分片上，其他 11 个分片都闲着，这样不仅浪费资源，很可能你 3 月那个分片根本抗不住几乎全部的并发请求。这个问题就是“热点问题”。

也就是说，我们希望并发请求和数据能均匀地分布到每一个分片上，尽量避免出现热点。这是选择分片算法时需要考虑的一个重要的因素。一般常用的分片算法就那么几种，刚刚讲到的按照时间范围分片的方法是其中的一种。

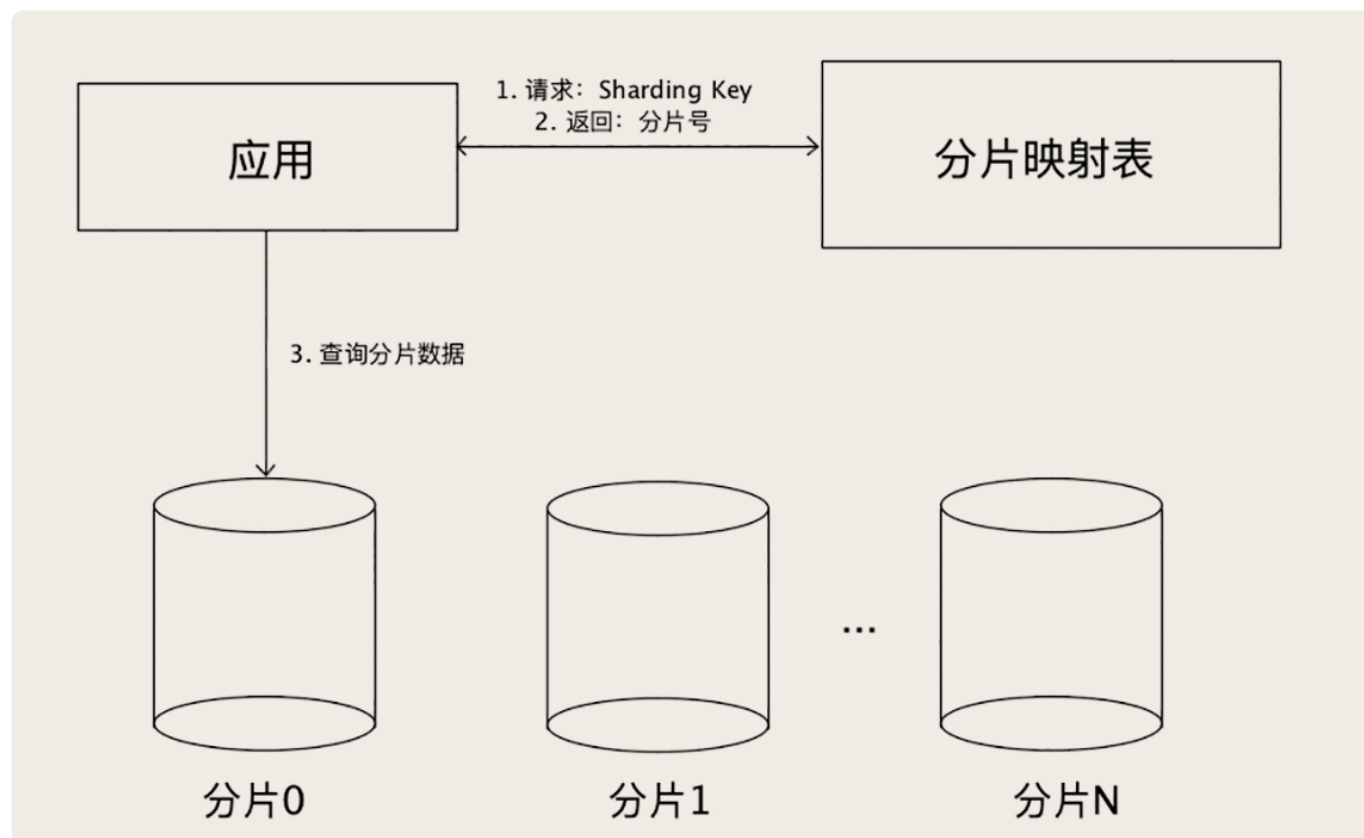
基于范围来分片容易产生热点问题，不适合作为订单的分片方法，但是这种分片方法的优点也很突出，那就是对查询非常友好，基本上只要加上一个时间范围的查询条件，原来该怎么查，分片之后还可以怎么查。范围分片特别适合那种数据量非常大，但并发访问量不大的 ToB 系统。比如说，电信运营商的监控系统，它可能要采集所有人手机的信号质量，然后做一些分析，这个数据量非常大，但是这个系统的使用者是运营商的工作人员，并发量很少。这种情况下就很适合范围分片。

一般来说，订单表都采用更均匀的哈希分片算法。比如说，我们要分 24 个分片，选定了 Sharding Key 是用户 ID，那我们决定某个用户的订单应该落到那个分片上的算法是，拿用户 ID 除以 24，得到的余数就是分片号。这是最简单的取模算法，一般就可以满足大部分要求了。当然也有一些更复杂的哈希算法，像一致性哈希之类的，特殊情况下也可以使用。

需要注意的一点是，哈希分片算法能够分得足够均匀的前提条件是，用户 ID 后几位数字必须是均匀分布的。比如说，你在生成用户 ID 的时候，自定义了一个用户 ID 的规则，最后一位 0 是男性，1 是女性，这样的用户 ID 哈希出来可能就没那么均匀，可能会出现热点。

还有一种分片的方法：查表法。查表法其实就是没有分片算法，决定某个 Sharding Key 落在哪个分片上，全靠人为来分配，分配的结果记录在一张表里面。每次执行查询的时候，先去表里查一下要找的数据在哪个分片中。

查表法的好处就是灵活，怎么分都可以，你用上面两种分片算法都没法分均匀的情况下，就可以用查表法，人为地来把数据分均匀了。查表法还有一个特好的地方是，它的分片是可以随时改变的。比如我发现某个分片已经是热点了，那我可以把这个分片再拆成几个分片，或者把这个分片的数据移到其他分片中去，然后修改一下分片映射表，就可以在线完成数据拆分了。



但你需要注意的是，分片映射表本身的数据不能太多，否则这个表反而成为热点和性能瓶颈了。查表法相对其他两种分片算法来说，缺点是需要二次查询，实现起来更复杂，性能上也稍微慢一些。但是，分片映射表可以通过缓存来加速查询，实际性能并不会慢很多。

小结

对 MySQL 这样的单机数据库来说，分库分表是应对海量数据和高并发的最后一招，分库分表之后，将会对数据查询有非常大的限制。

分多少个库需要用并发量来预估，分多少表需要用数据量来预估。选择 Sharding Key 的时候，一定要能兼容业务最常用的查询条件，让查询尽量落在一个分片中，分片之后无法兼容的查询，可以把数据同步到其他存储中去，来解决这个问题。

我们常用三种分片算法，范围分片容易产生热点问题，但对查询更友好，适合适合并发量不大的场景；哈希分片比较容易把数据和查询均匀地分布到所有分片中；查表法更灵活，但性能稍差。

对于订单表进行分库分表，一般按照用户 ID 作为 Sharding Key，采用哈希分片算法来均匀分布用户订单数据。为了能支持按订单号查询的需求，需要把用户 ID 的后几位放到订单号中去。

最后还需要强调一下，我们这节课讲的这些分片相关的知识，不仅仅适用于 MySQL 的分库分表，你在使用其他分布式数据库的时候，一样会遇到如何分片、如何选择 Sharding Key 和分片算法的问题，它们的原理都是一样的，所以我们讲的这些方法也都是通用的。

思考题

课后请你想一下，把订单表拆分之后，那些和订单有外键关联的表，该怎么处理？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。


后端存储实战课

类电商平台存储技术应用指南

李玥

京东零售计算存储平台部资深架构师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 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。

上一篇 14 | 订单数据越来越多，数据库越来越慢该怎么办？

精选留言 (10)

 写留言



李玥 置顶

2020-04-01

Hi，我是李玥。

这里回顾一下上节课的思考题：

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...
展开



虚竹

2020-03-31

老师好，订单id总共18位，10-14位放用户id的后几位，当用户要查询自己的所有订单时怎么查呢？这里没太明白

展开

 2

 1



正在减肥的胖籽。

2020-03-31

老师您好，有几个问题需要请教你下：

1.运营后台查询数据，肯定就是查询很多数据，不止是单个用户的数据。京东怎么去处理运营后台的数据呢？是把数据同步到ES中还是？



gfgf

2020-04-01

用户id和订单id关联这块，订单中取用户id的几位来做分区分表的依据，这个做法是否会造成数据量的不均衡；

看到个人淘宝订单是嵌入了用户id的后几位，但是京东订单没有发现规律，这块能否请老师以淘宝和京东订单为例，做一个更深入的介绍

展开 ∨



靠人品去赢

2020-04-01

我觉得是外联的表数据量是不是也很大，如果外联的表并发小，数量少，其实不必要管他更不用也对他进行分库分表，因为从维护角度讲，能不拆就不拆，毕竟这是最后一步。



每天晒白牙

2020-04-01

分库分表的原则

能不拆就不拆，能少拆就少拆

分库分表的目的

1.数据量大查询慢的问题...

展开 ∨



一步

2020-03-31

分库分表使用的场景是不一样的，分表是因为数据量比较大查询较慢才进行的，分库是因为单库的性能无法满足要求才进行的

展开 ∨





sami

2020-03-31

感觉Redis Cluster的分片规则有点像查表法

展开 ▾



饭团

2020-03-31

老师您好，您提到的把用户id放到订单id中，是说如果用订单id去做查询，就把第10-14位取出来，之后再用这部分去查询是吗？

展开 ▾

12



Mq

2020-03-31

关联表可以冗余用户ID字段，跟订单表的都在一个库里吗，这样用户维度的交易都在一个库事物里面，大部分的关联查询也是在一个库里。

老师查找法有具体例子吗，我们之前也有过表按hash分后数据极度不均

展开 ▾

