




# Database

Oliver

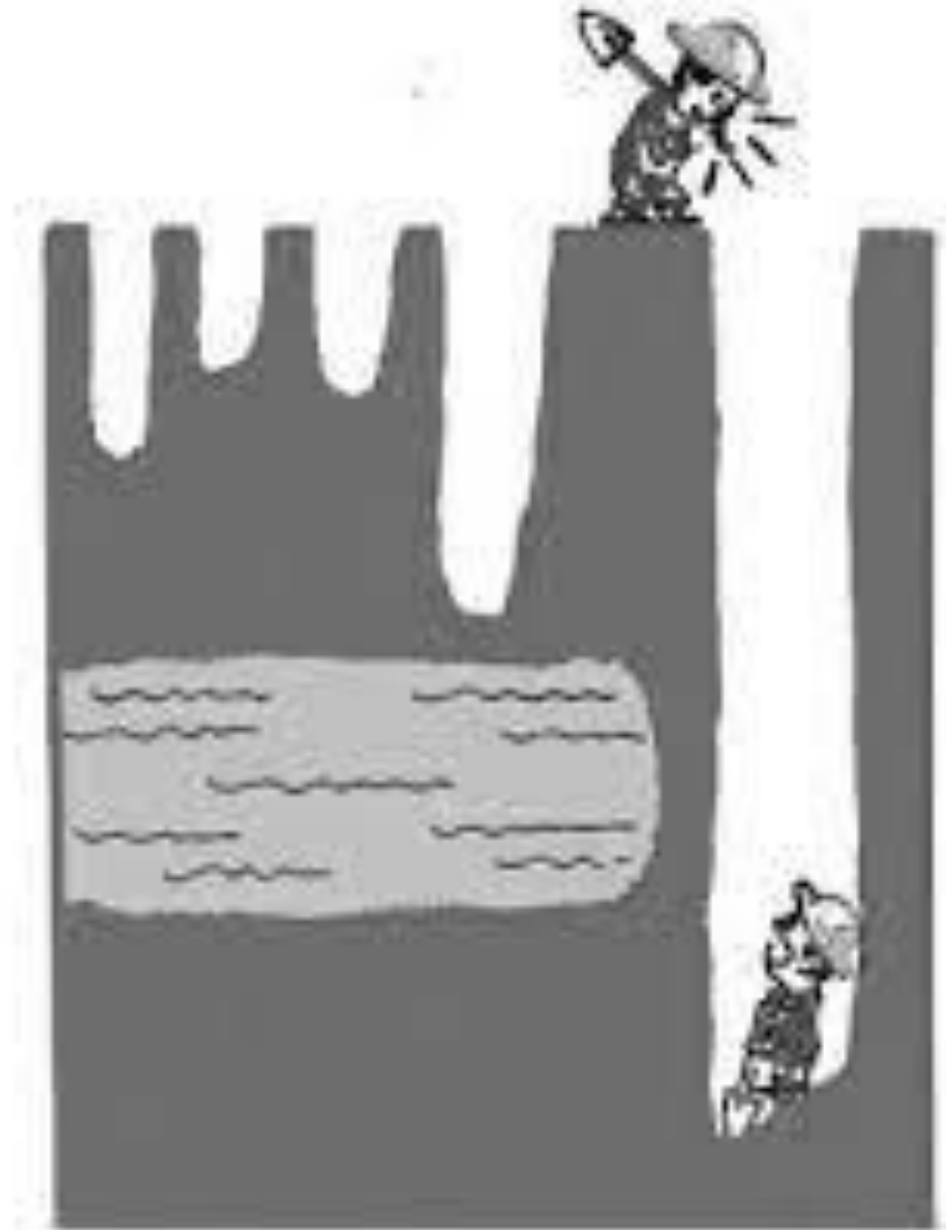


# 纯属抛砖引玉

- 欢迎大牛补充！
- Disclaimer: 不做商业用途。本整理纯属用于非盈利学习。部分资料来源于互联网。

# 《挖井》

- 有一位年轻人问一位大师：“大师，我们在人间为人处世太苦了，有什么方法可以教我吗？”大师就说：“两个人在挖井，一个人很会动脑筋，而另一个人很笨。两人挖了两米深，都没看到水，笨人继续挖，聪明精干的人就换了个地方挖，最后笨人挖到了水，而聪明人换来换去一无所获。”
- 大师继续说故事：“聪明人经过数次尝试，发现了一个水源；而笨人埋头苦干越挖越深，本来挖到的一些水并不是水源，所以他付出很多，但最终没有找到水源。”
- 大师笑笑，说：“这个故事还没讲完，这两个人虽然竭尽全力，聪明人换了很多地方，笨人拼命往下挖，两个人还是没有挖到大水源。”年轻人说：“那我们做人有没有处世的准则和哲学？”



甲：收吧，这里根本没有水！  
乙：挖不出水，我绝不收兵！

# T-Shaped Developer/ Engineer

Agile Notes

## Benefits of adopting T-Shape mindset:

- Development team members strengthen existing skills and learn new skills
- Remove silos within the team, bridge gaps and be cross-functional
- Increase team utilization and velocity by not playing the “waiting game”

Analyst	Programmer	Test Engineer	Web Designer	System Engineer
Write Executable Documents	Write Unit Test Code (xUnit)	Write Automated Tests	UX Design	DevOps
Requirements Engineering	Write Production Code	Functional Testing	Java Script, HTML, CSS, LeSS	Python, Perl, Go, shell
Write User Manuals	Design System Architecture, DB	Write Test Plan	Image, Icon, Logo Design	System and OS
...	...	...	...	...

## Ideal Scrum Developer



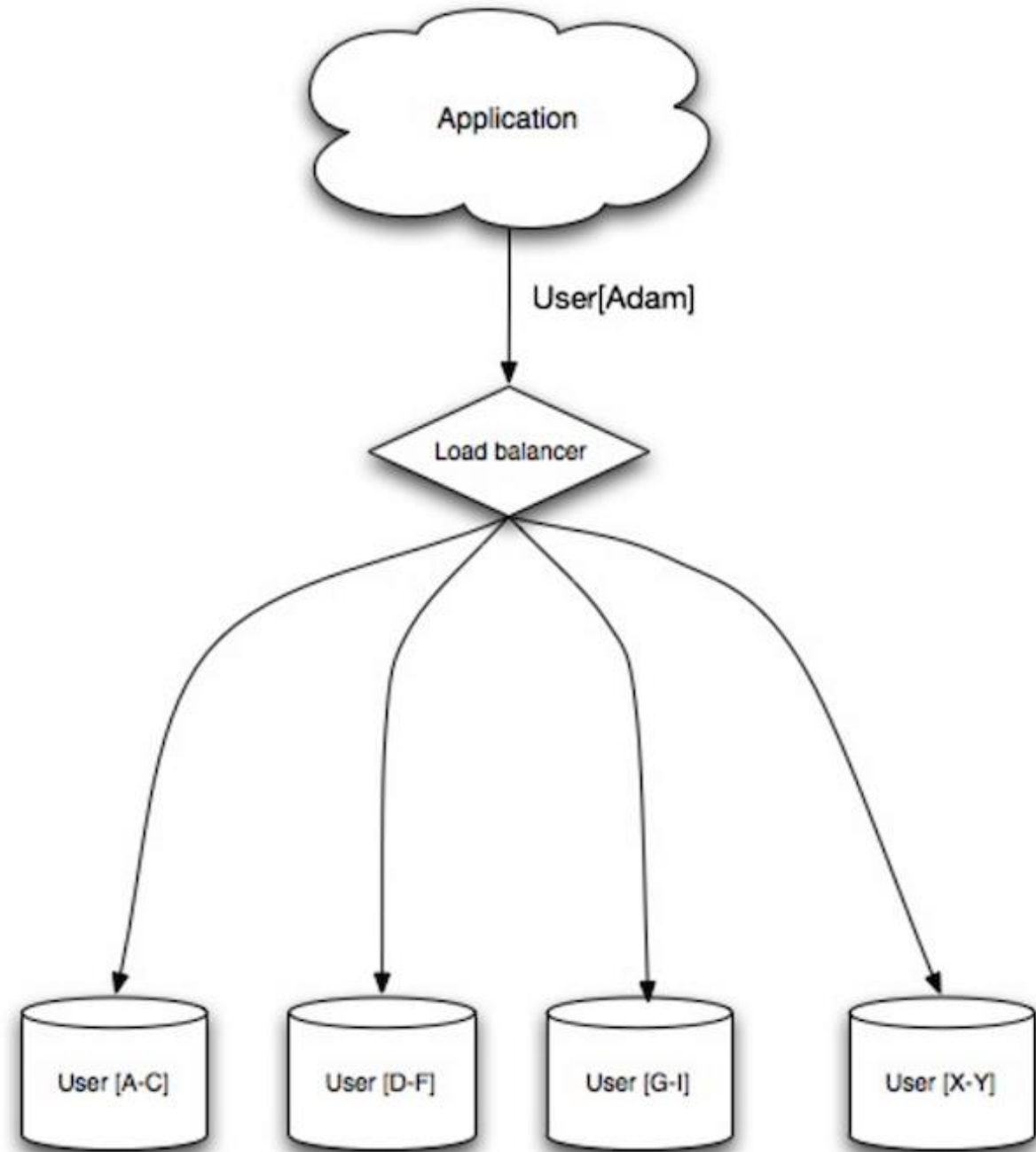
# 延伸阅读Big Tech Blogs

- [The Airbnb Tech Blog](#)
- [Uber Engineering](#)
- [Netflix Tech Blog](#)
- [Dropbox Tech Blog](#)
- [Twitter Engineering](#)
- [Facebook Tech Blog](#)
- [LinkedIn Engineering Blog](#)
- [Google Developers](#)
- [Mozilla Tech Blog](#)
- [AWS Architecture Blog](#)
- [All Things Distributed AWS CTO's Blog](#)



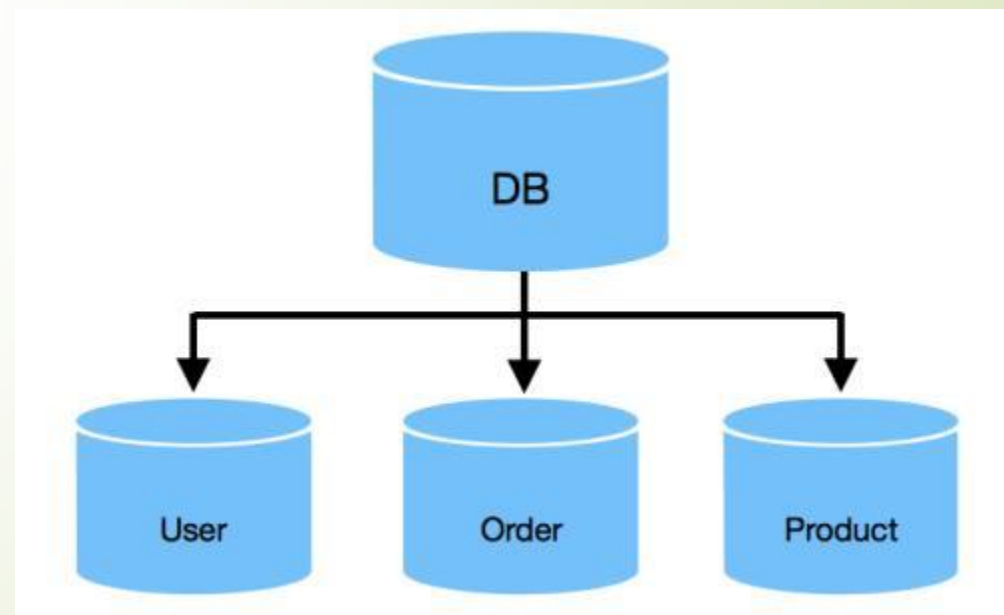
# Sharding

- Sharding distributes data across different databases such that each database can only manage a subset of the data. Taking a users database as an example, as the number of users increases, more shards are added to the cluster.
- Similar to the advantages of federation, sharding results in less read and write traffic, less replication, and more cache hits. Index size is also reduced, which generally improves performance with faster queries. If one shard goes down, the other shards are still operational, although you'll want to add some form of replication to avoid data loss. Like federation, there is no single central master serializing writes, allowing you to write in parallel with increased throughput.
- Common ways to shard a table of users is either through the user's last name initial or the user's geographic location.



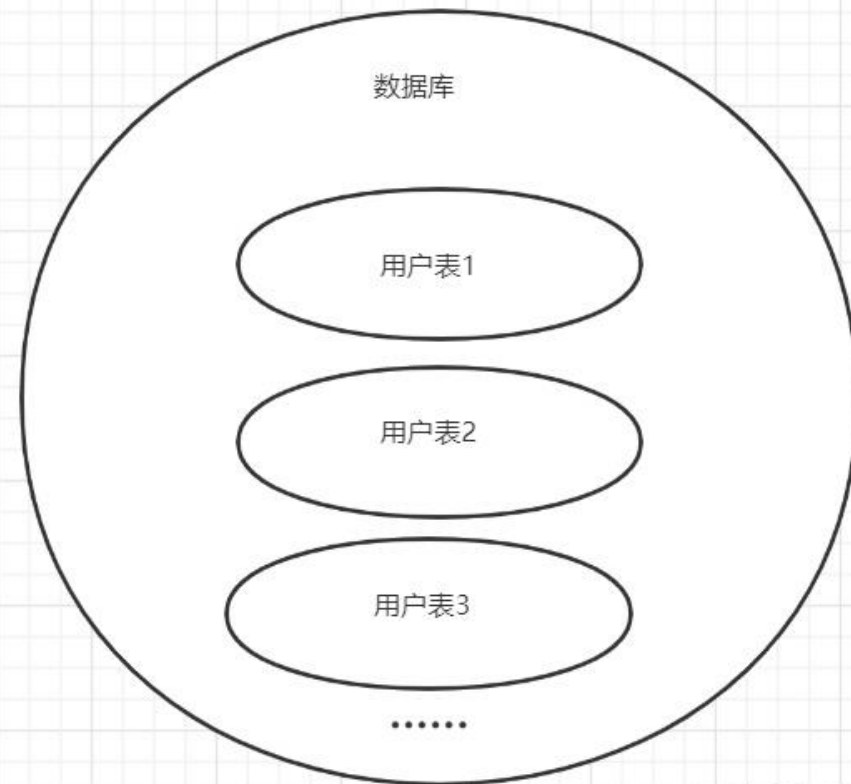
# 分库分表

- 在 Internet 项目中比较常用到的关系型数据库是 MySQL，随着用户和业务的增长，传统的单库单表模式难以满足大量的业务数据存储以及查询，单库单表中大量的数据会使写入、查询效率非常之慢，此时应该采取分库分表策略来解决。



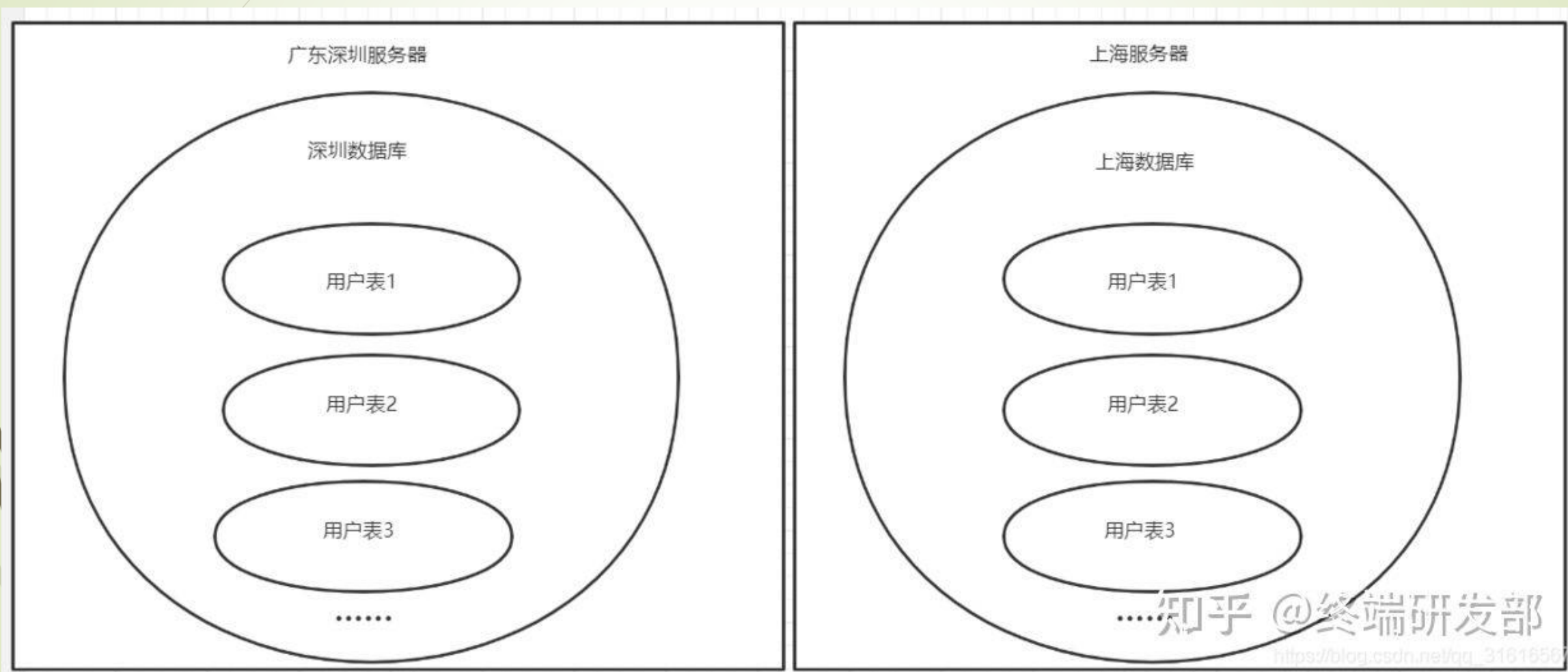
# 一、业务场景介绍

- 数据库中有用户表。用户会非常多，并且要实现：我们先看传统的分库分表方式
- Can you do better?



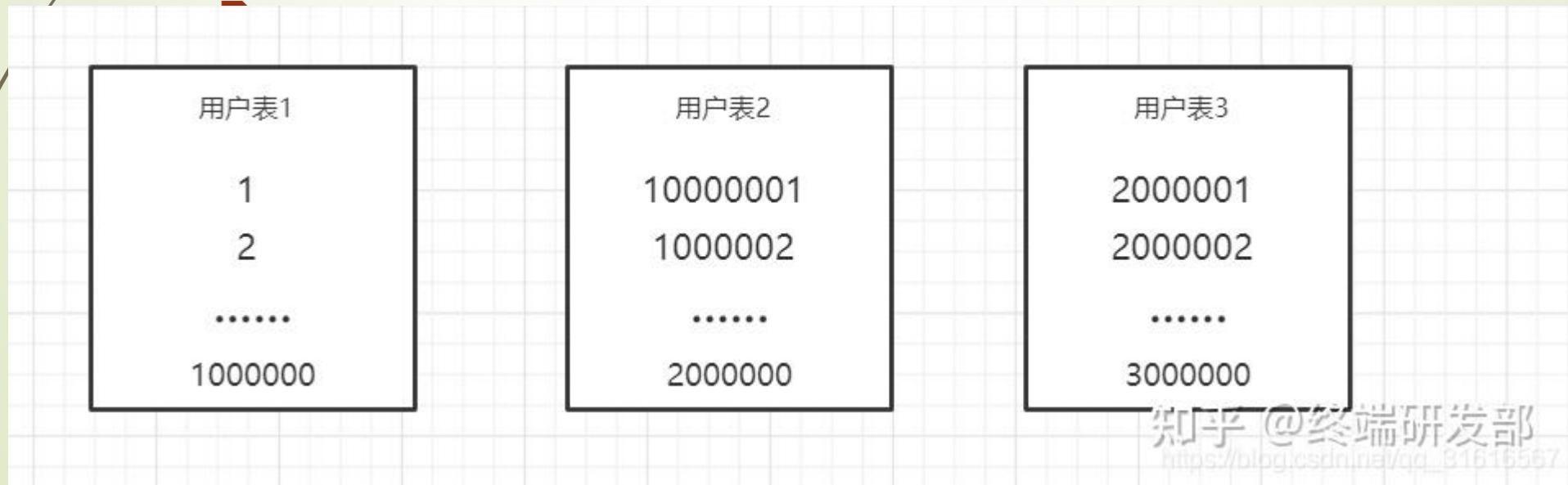



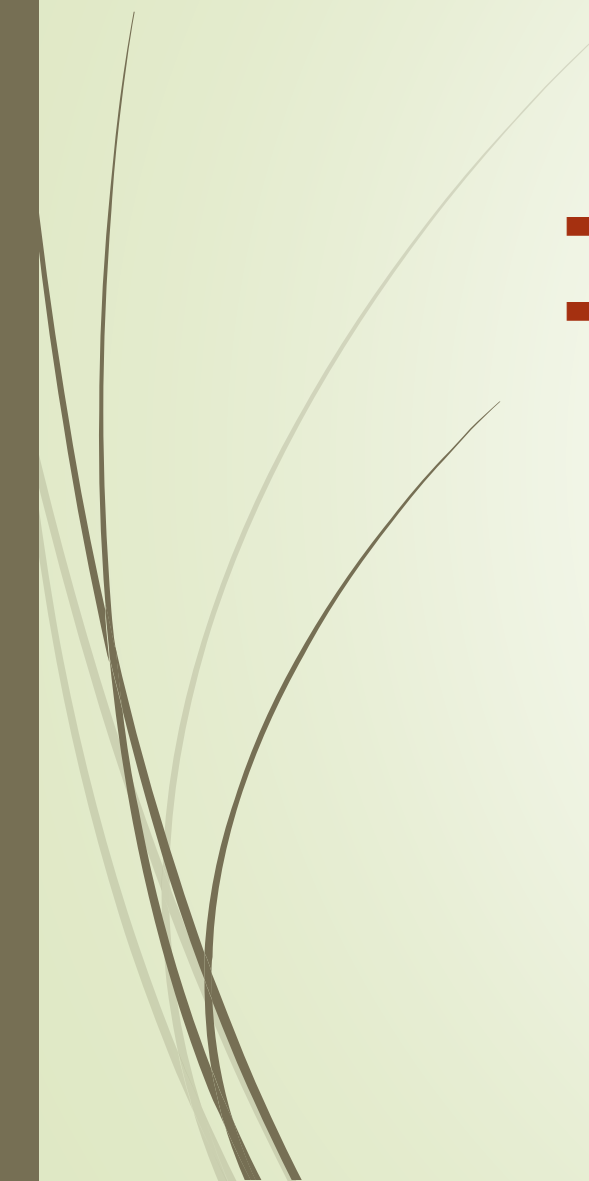
当然还有些小伙伴知道按照省份/地区或一定的业务关系进行数据库拆分



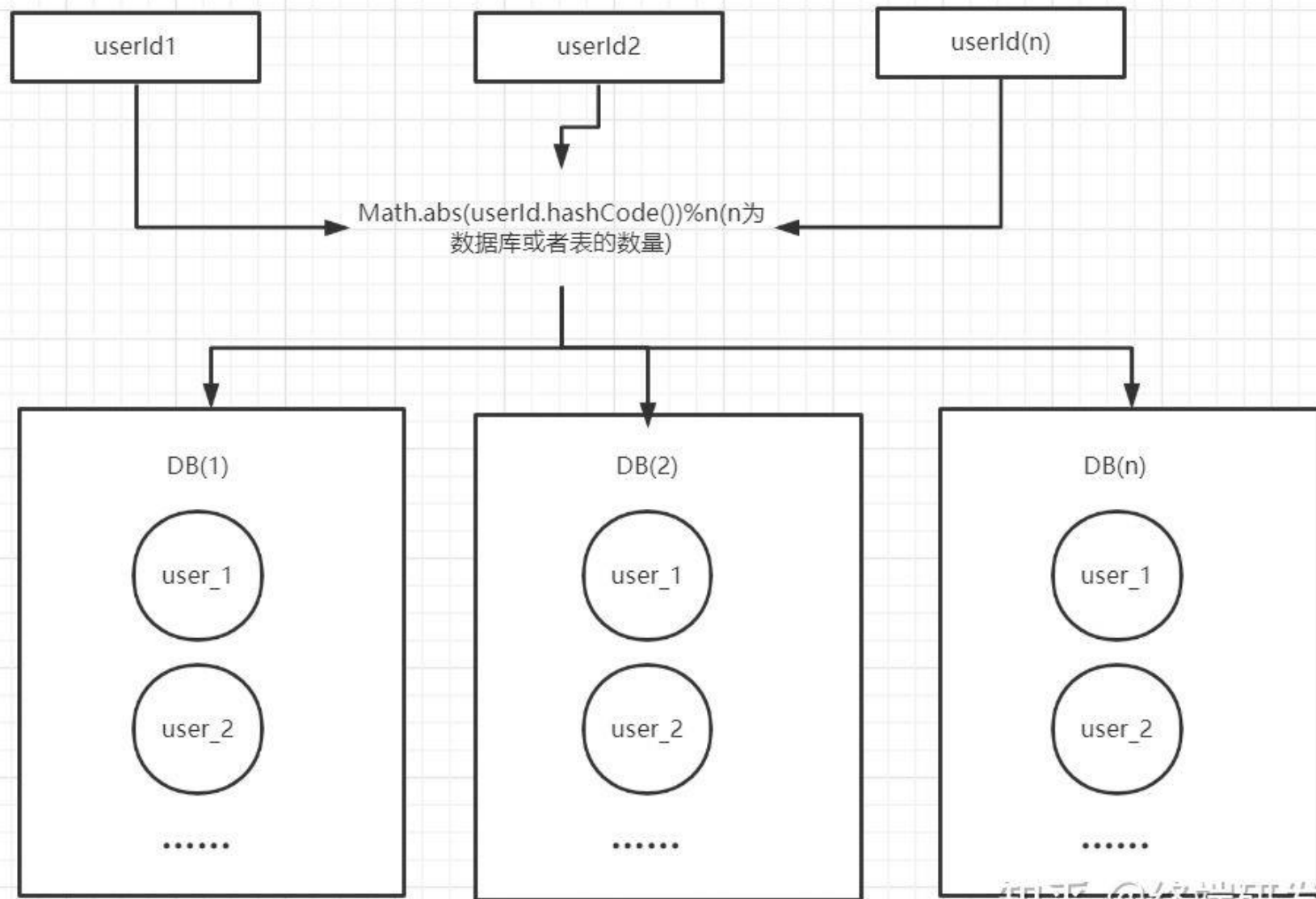
## 二、水平分库分表方法1.RANGE方法



- 第一种方法可以指定一个数据范围来进行分表，例如从1~1000000，1000001-2000000，使用一百万一张表的方式，如下图所示
- 当然这种方法需要维护表的ID，特别是分布式环境下，这种分布式ID,在不使用第三方分表工具的情况下，建议使用Redis，Redis的incr操作可以轻松的维护分布式的表ID。



- 
- 
- **RANGE**方法优点：扩容简单，提前建好库、表就好
  - **RANGE**方法缺点：大部分读和写都访会问新的数据，有IO瓶颈，这样子造成新库压力过大，不建议采用。

## 2.HASH取模



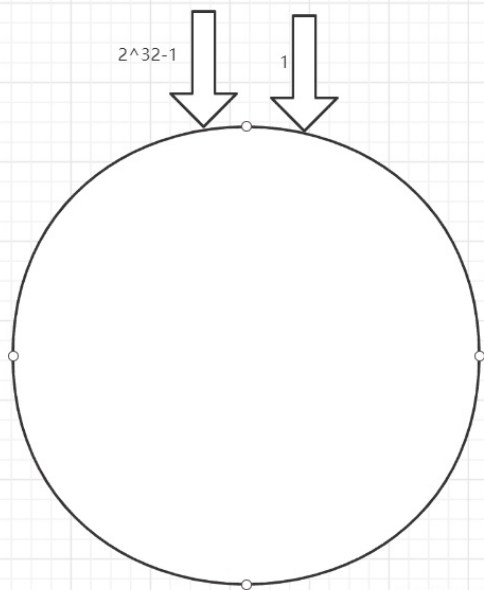
- 
- 
- **HASH取模方法优点：** 能保证数据较均匀的分散落在不同的库、表中，减轻了数据库压力
  - **HASH取模方法缺点：** 扩容麻烦、迁移数据时每次都需要重新计算hash值分配到不同的库和表



### 3.一致性HASH

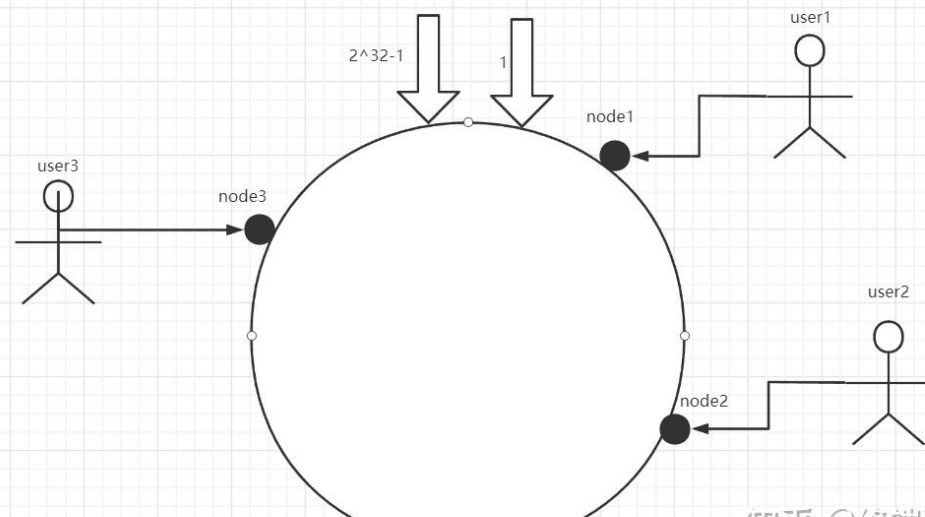
- 通过HASH取模也不是最完美的办法，那什么才是呢？
- 使用一致性HASH算法能完美的解决问题
- 普通HASH算法：
- 普通哈希算法将任意长度的二进制值映射为较短的固定长度的二进制值，这个小的二进制值称为哈希值。哈希值是一段数据唯一且极其紧凑的数值表示形式。
- 普通的hash算法在分布式应用中的不足：在分布式的存储系统中，要将数据存储到具体的节点上，如果我们采用普通的hash算法进行路由，将数据映射到具体的节点上，如 $key \% n$ ， $key$ 是数据的key， $n$ 是机器节点数，如果有一个机器加入或退出集群，则所有的数据映射都无效了，如果是持久化存储则要做数据迁移，如果是分布式缓存，则其他缓存就失效了。

# 一致性HASH算法：

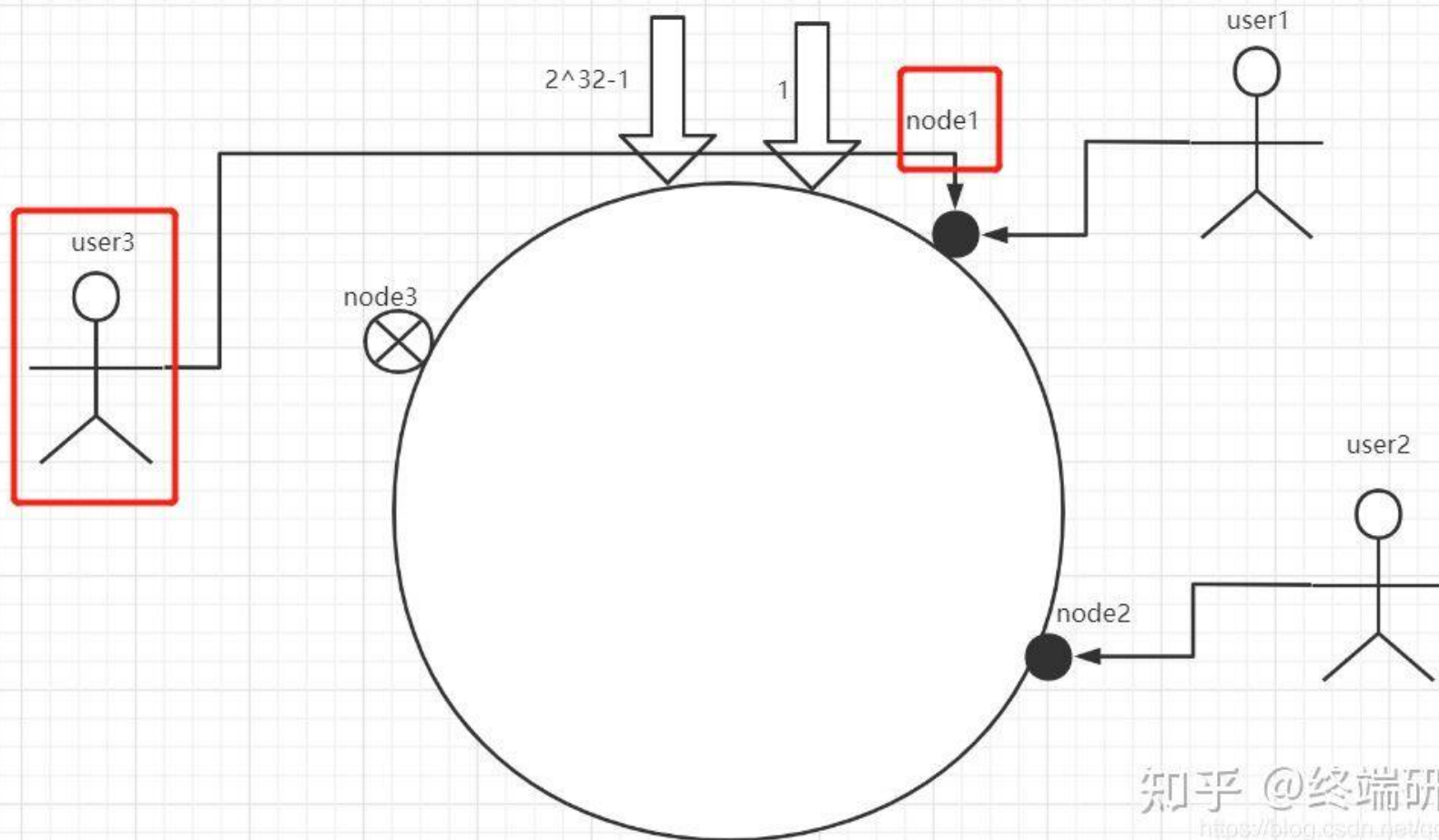


[https://blog.csdn.net/qq\\_34374651/article/details/80000000](https://blog.csdn.net/qq_34374651/article/details/80000000)

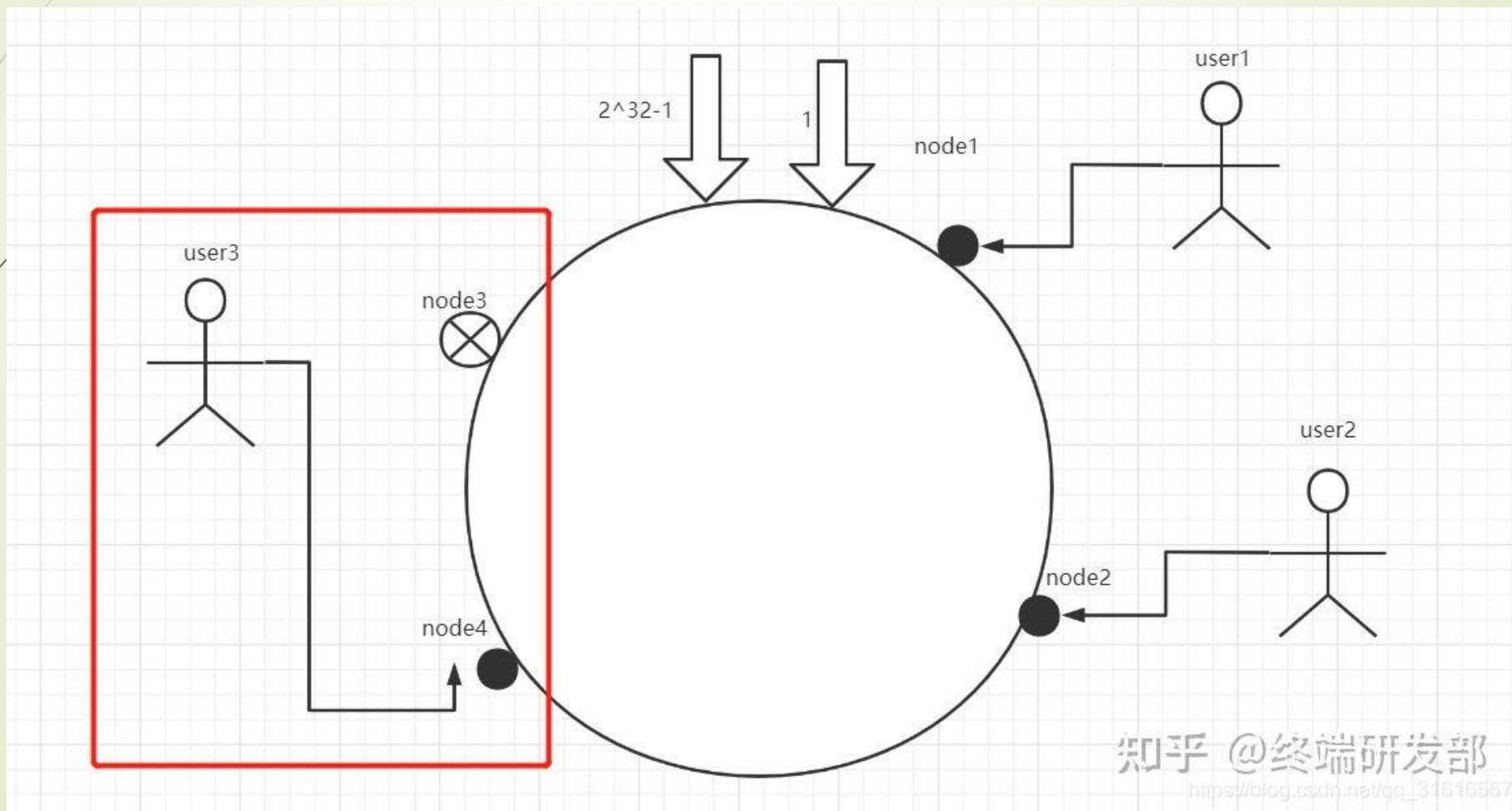
- 一致性HASH算法：按照常用的hash算法来将对应的key哈希到一个具有 $2^{32}$ 次方个节点的空间中，即 $0 \sim (2^{32})-1$ 的数字空间中。现在我们可以将这些数字头尾相连，想象成一个闭合的环形，如下图所示。
- 这个圆环首尾相连，那么假设现在有三个数据库服务器节点node1、node2、node3三个节点，每个节点负责自己这部分的用户数据存储，假设有用户user1、user2、user3，我们可以对服务器节点进行HASH运算，假设HASH计算后，user1落在node1上，user2落在node2上，user3落在node3上。




OK，现在咱们假设node3节点失效了



再假设新增了节点node4





# 平衡性???

- 当然还有一个问题还需要解决，那就是平衡性。从图我们可以看出，当服务器节点比较少的时候，会出现一个问题，就是此时必然造成大量数据集中到一个节点上面，极少数数据集中到另外的节点上面。





# 虚拟节点?!

- 为了解决这种数据倾斜问题，一致性哈希算法引入了虚拟节点机制，即对每一个服务节点计算多个哈希，每个计算结果位置都放置一个节点，称为虚拟节点。具体做法可以先确定每个物理节点关联的虚拟节点数量，然后在ip或者主机名后面增加编号。例如上面的情况，可以为每台服务器计算三个虚拟节点，于是可以分别计算“node 1-1”、“node 1-2”、“node 1-3”、“node 2-1”、“node 2-2”、“node 2-3”、“node 3-1”、“node 3-2”、“node 3-3”的哈希值，这样形成九个虚拟节点
- 例如user1定位到node 1-1、node 1-2、node 1-3上其实都是定位到node1这个节点上，这样能够解决服务节点少时数据倾斜的问题，当然这个虚拟节点的个数不是说固定三个或者至多、至少三个，这里只是一个例子，具体虚拟节点的多少，需要根据实际的业务情况而定。

# 一致性HASH方法优点：

- ▶ 通过虚拟节点方式能保证数据较均匀的分散落在不同的库、表中，并且新增、删除节点不影响其他节点的数据，高可用、容灾性强。
- ▶ 一致性取模方法缺点： 嗯，比起以上两种，可以认为没有。
- ▶

# 为什么分库分表？？？

- 随着用户量的激增和时间的堆砌，存在数据库里面的数据越来越多，此时的数据库就会产生瓶颈，出现资源报警、查询慢等场景。
- 先单机数据库所能承载的连接数、I/O及网络的吞吐等都是有限的，所以当并发量上来了之后，数据库就渐渐顶不住了。
- 再则，如果单表的数据量过大，查询的性能也会下降。因为数据越多 B+ 树就越高，树越高则查询 I/O 的次数就越多，那么性能也就越差。

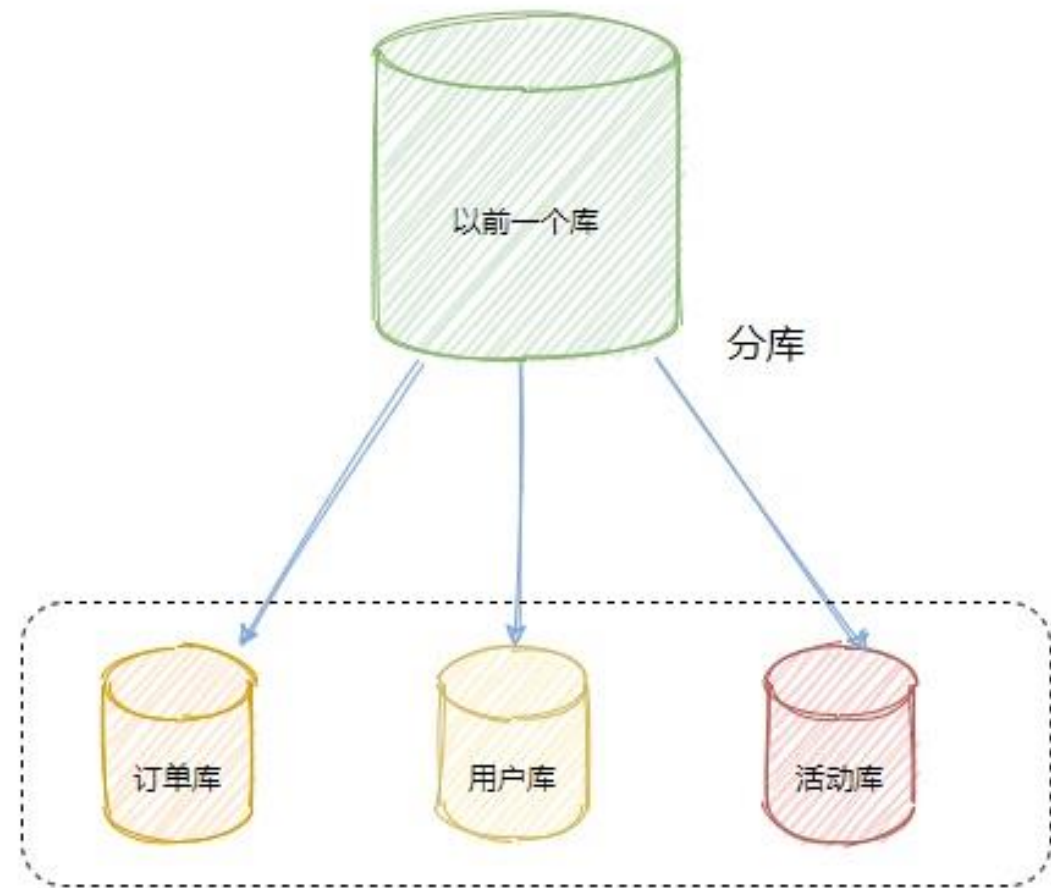


# 分库分表

- 把以前存在一个数据库实例里的数据拆分成多个数据库实例，部署在不同的服务器中，**这是分库。**
- 把以前存在一张表里面的数据拆分成多张表，**这是分表。**
- 一般而言：
  - 分表：是为了解决由于单张表数据量多大，而导致查询慢的问题。大致三、四千万行数据就得拆分，不过具体还是得看每一行的数据量大小，有些字段都很小的可能支持更多行数，有些字段大的可能一千万就顶不住了。
  - 分库：是为了解决服务器资源受单机限制，顶不住高并发访问的问题，把请求分配到多台服务器上，降低服务器压力。

# 怎么分库的？

- 一般分库都是按照业务划分的，比如订单库、用户库等等。
- 有时候会针对一些特殊的库再作切分，比如一些活动相关的库都做了拆分。
- 因为做活动的时候并发可能会比较高，怕影响现有的核心业务，所以即使有关联也会单独做拆分。





# 那你觉得分库会带来什么问题呢？

- 首先是事务的问题。
  - 我们使用关系型数据库，有很大一点在于它保证事务完整性。
  - 而分库之后单机事务就用不上了，必须使用分布式事务来解决，而分布式事务基本的都是残缺的(我之前文章把分布式事务汇总了一波，后台搜索分布式事务就有了)。
  - 这是很重要的一点需要考虑。
- 连表 JOIN 问题
  - 在一个库中的时候我们还可以利用 JOIN 来连表查询，而跨库了之后就无法使用 JOIN 了。
  - 此时的解决方案就是在业务代码中进行关联，也就是先把一个表的数据查出来，然后通过得到的结果再去查另一张表，然后利用代码来关联得到最终的结果。
  - 这种方式实现起来稍微比较复杂，不过也是可以接受的。
  - 还有可以适当的冗余一些字段。比如以前的表就存储一个关联 ID，但是业务时常要求返回对应的 Name 或者其他字段。这时候就可以把这些字段冗余到当前表中，来去除需要关联的操作。

# 那你们怎么分表的？

➡ 分表其实有两种：

- 垂直分表
- 水平分表

# 垂直分表

- 垂直分表就是把一些不常用的大字段剥离出去。
- 像上面的例子：用户名是很常见的搜索结果，性别和年龄占用的空间又不大，而地址和个人简介占用的空间相对而言就较大，我们都知道一个数据页的空间是有限的，把一些无用的数据拆分出去，一页就能存放更多行的数据。
- 内存存放更多有用的数据，就减少了磁盘的访问次数，性能就得到提升。

在这里垂直一刀切开

ID	NAME	SEX	AGE	ADDRESS	INTRODUCE	....
1						
2						
3						
4						
5						

垂直分表



ID	NAME	SEX	AGE
1			
2			
3			
4			
5			

ID	ADDRESS	INTROUCE	....
1			
2			
3			
4			
5			

# 水平分表

- 水平分表，则是因为一张表内的数据太多了，上文也提到了数据越多 B+ 树就越高，访问的性能就差，所以进行水平拆分。
- 其实不管这些，浅显的理解下，在一百个数据里面找一个数据快，还是一万个数据里面找一个数据快？
- 即使有索引，那厚的书目录多，翻目录也慢~





## 5.1、非partition key的查询问题（水平分库分表，拆分策略为常用的hash法）

- 所以设计上得考虑查询的条件来作为 Sharding-Key。
- 举个常常会被问的订单表 Sharding-Key 例子。
- 你想着查找订单的时候会通过订单号去找，所以应该利用订单 ID 来作为 Sharding-Key。
- 但是你想想，你打开外卖软件想查找你的历史订单的时候，你是没有订单 ID 的，你只有你的 UserID，那此时只能把所有子表都通过 UserID 遍历一遍，这样效率就很低了！
- 所以你想着那用 UserID 来作为 Sharding-Key 吧！
- 但是，商家呢？商家肯定关心自己今天卖了多少单，所以他也要查找订单，但他只有自己的商家 ID，所以如果要查询订单，只能把所有子表都通过商家 ID 遍历一遍，这样效率就很低了！
- 所以 Sharding-Key 是满足不了所有查询需求的，只能曲线救国。
- 一般做法就是冗余数据。
- 将订单同步到另一张表中给商家使用，这个表按商家 ID 来作为 Sharding-Key，也可以将数据同步到 ES 中。一般而言这里的数据同步都是异步处理，不会影响正常流程。



- 
- 
- 5.1.1 端上除了partition key只有一个非partition key作为条件查询
    - 映射法
  - 5.1.2 端上除了partition key不止一个非partition key作为条件查询
    - 冗余法
  - 5.1.3 后台除了partition key还有各种非partition key组合条件查询
    - NoSQL法（ES等）。
    - 冗余法
    -










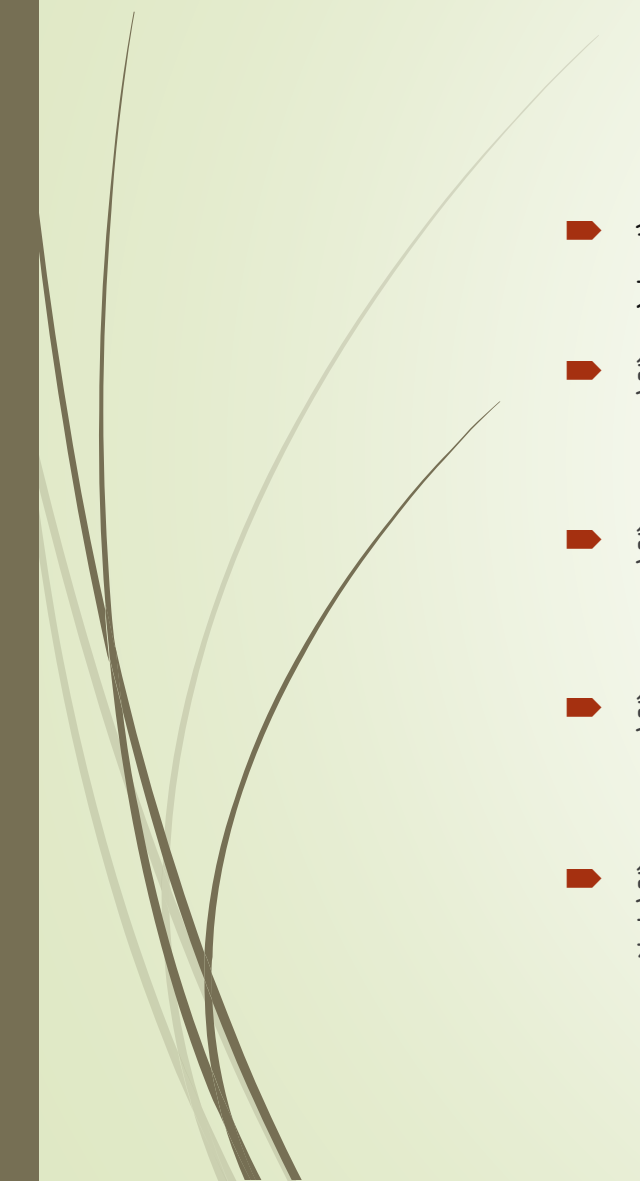




- 
- <https://www.zhihu.com/question/459955079/answer/1949098113>
  - <https://www.huaweicloud.com/articles/8ac1dc6dfe005bdaf76f7d1d2136276a.html>

# MySQL 分库分表方案



- 
- 
- 今天有人问到了数据库分库分表，他们都说数据库达到了瓶颈，需要重构，但是毫无头绪，现在做个概念总结（期待后期的实操吗？）会从以下几个方面说起：
  - 第一部分：实际网站发展过程中面临的问题。
  - 第二部分：有哪几种切分方式，垂直和水平的区别和适用面。
  - 第三部分：目前市面有的一些开源产品，技术，它们的优缺点是什么。
  - 第四部分：可能是最重要的，为什么不建议水平分库分表！？这能让你能在规划前期谨慎的对待，规避掉切分造成的问题。



# 名词解释



库：database；表：table；分库分表：sharding



# 数据库架构演变

- 刚开始我们只用单机数据库就够了，随后面对越来越多的请求，我们将数据库的写操作和读操作进行分离，使用多个从库副本（**Slaver Replication**）负责读，使用主库（**Master**）负责写，从库从主库同步更新数据，保持数据一致。架构上就是数据库主从同步。从库可以水平扩展，所以更多的读请求不成问题。

但是当用户量级上来后，写请求越来越多，该怎么办？加一个**Master**是不能解决问题的，因为数据要保存一致性，写操作需要2个**master**之间同步，相当于是重复了，而且更加复杂。

这时就需要用到分库分表（**sharding**），对写操作进行切分。

# 分库分表前的问题

- 任何问题都是太大或者太小的问题，我们这里面对的数据量太大的问题。
- **用户请求量太大**  
因为单服务器TPS，内存，IO都是有限的。解决方法：分散请求到多个服务器上；其实用户请求和执行一个sql查询是本质是一样的，都是请求一个资源，只是用户请求还会经过网关，路由，http服务器等。
- **单库太大**  
单个数据库处理能力有限；单库所在服务器上磁盘空间不足；单库上操作的IO瓶颈 解决方法：切分成更多更小的库
- **单表太大**  
CRUD都成问题；索引膨胀，查询超时 解决方法：切分成多个数据集更小的表。

# 分库分表的方式方法

- 一般就是垂直切分和水平切分，这是一种结果集描述的切分方式，是物理空间上的切分。
- 我们从面临的问题，开始解决，阐述：首先是用户请求量太大，我们就堆机器搞定（这不是本文重点）。
- 然后是单个库太大，这时我们要看是因为表多而导致数据多，还是因为单张表里面的数据多。如果是因为表多而数据多，使用垂直切分，根据业务切分成不同的库。
- 如果是因为单张表的数据量太大，这时要用水平切分，即把表的数据按某种规则切分成多张表，甚至多个库上的多张表。
- 分库分表的顺序应该是**先垂直分，后水平分**。因为垂直分更简单，更符合我们处理现实世界问题的方式。

# 分库分表方案产品

- 目前市面上的分库分表中间件相对较多,
- 其中基于代理方式的有MySQL Proxy和Amoeba,
- 基于Hibernate框架的是Hibernate Shards
- 基于jdbc的有当当sharding-jdbc,
- ~~其他基于mybatis的类似maven插件式的ibatis-template类的Cobar-Client。~~

## 分库分表技术调研



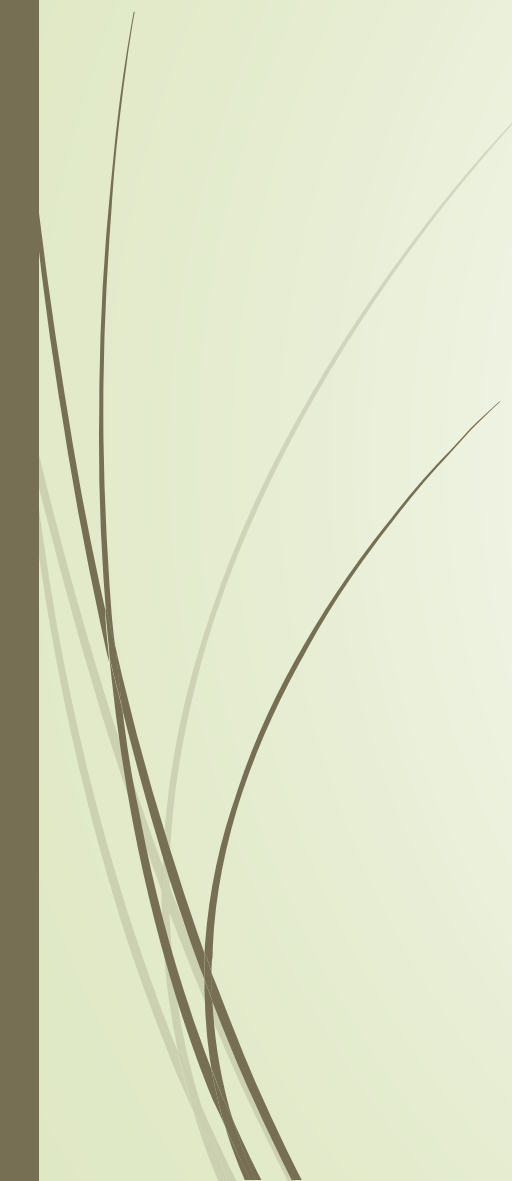






# Q & A

➡ 踊跃发言





# Thank You!

- References:
- <https://github.com/donnemartin/system-design-primer>
- <https://zhuanlan.zhihu.com/p/303205010>