# 一．数据切分

关系型数据库本身比较容易成为系统瓶颈，单机存储容量、连接数、处理能力都有限。当单表的数据量达到1000W或100G以后，由于查询维度较多，即使添加从库、优化索引，做很多操作时性能仍下降严重。此时就要考虑对其进行切分了，切分的目的就在于减少数据库的负担，缩短查询时间。

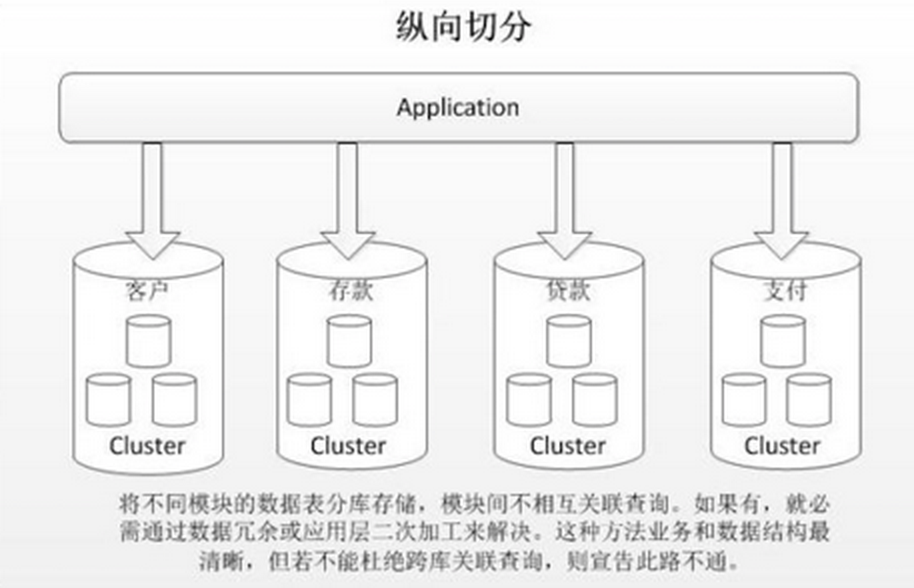
数据库分布式核心内容无非就是数据切分（Sharding），以及切分后对数据的定位、整合。数据切分就是将数据分散存储到多个数据库中，使得单一数据库中的数据量变小，通过扩充主机的数量缓解单一数据库的性能问题，从而达到提升数据库操作性能的目的。

数据切分根据其切分类型，可以分为两种方式：垂直（纵向）切分和水平（横向）切分

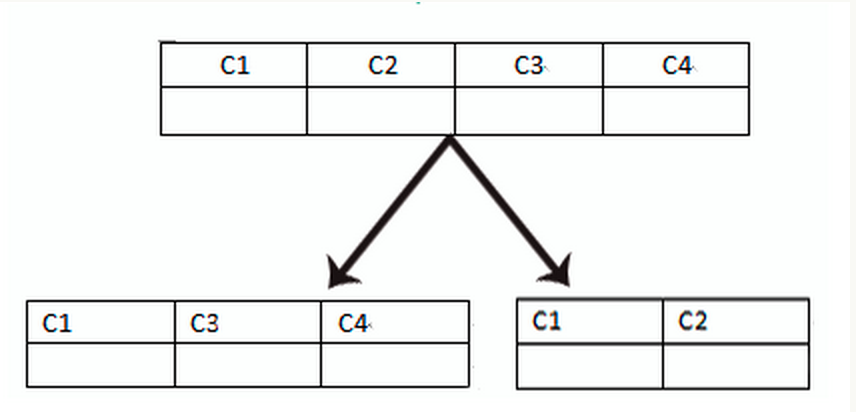
## 垂直切分

垂直切分通常分为垂直分库和垂直分表两种

垂直分库就是根据业务耦合性，将关联度低的不同表存储在不同的数据库中，如图：



垂直分表是基于数据库的 ‘列’进行，某个表字段较多，可以新建一张扩展表，将不经常用或字段长度较长的字段拆分出去到扩展表中，便于开发和维护，也未能避免跨页问题，mysql底层是通过数据页存储的，一条记录占用空间过大会导致跨页，造成额外的性能开销。数据库以行为单位将数据加载到内存 中，表中字段长度较短且频率较高，内存能加载更多的数据，命中率更高，减少磁盘IO，提升数据库性能。



### 垂直分区的优点：

1. 解决业务系统层面的耦合，业务清晰
2. 与微服务的治理类似，也能对不同业务的数据进行分级管理，维护，监控，扩展等
3. 高并发场景下，垂直切分一定程度的提升IO，数据库连接数，单机硬件资源的瓶颈

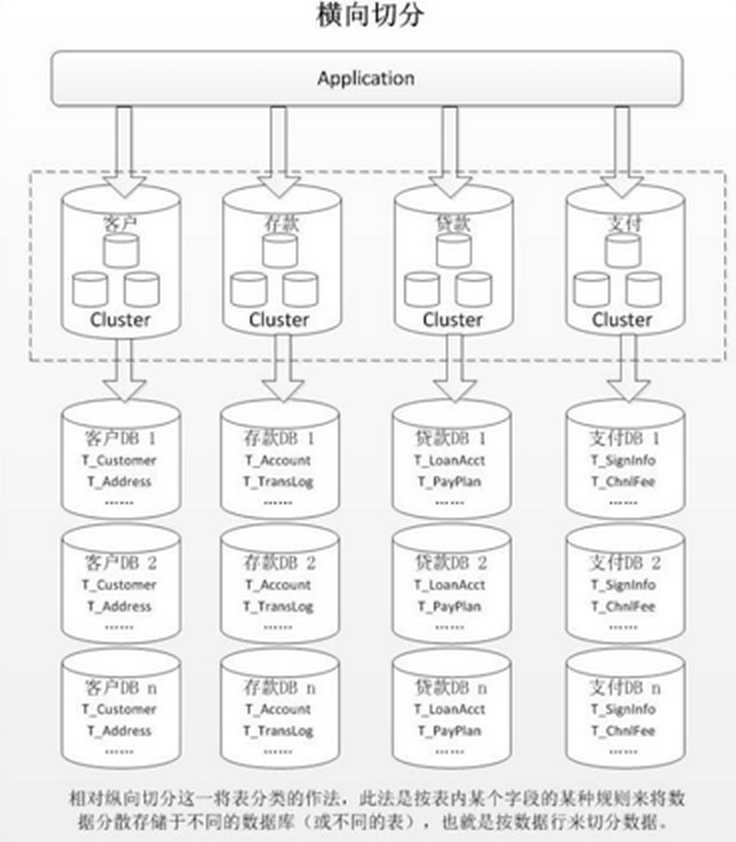
### 缺点：

1. 部分表无法join，只能通过接口聚合方式解决，提升了开发难度。
2. 分布式事务处理复杂
3. 依然攒在单表数据量过大的问题

## 水平切分

当一个运用难以再细粒度的垂直切分，或者切分后数据量行数巨大，存在单裤读写，存储性能瓶颈，这时候就需要进行水平切分。

水平切分分为库内分表和分库分表，根据表内数据内在的逻辑关系，将同一个表按不同的条件分散到多个数据库或多个表中，每个表中只包含一部分数据，从而使得单个表的数据量变小，达到分布式的效果，如下：



库内分表只解决了单一表数据量过大的问题，但没有将表分布到不同机器上的库上，对于减轻MYSQL数据库的压力来说帮助不是很大，大家还是竞争同一个物理机的CPU，内存，网络IO，做好通过分库分表来解决。

### 优点：

1. 不存在单库数据量过大，高并发的性能瓶颈，提升系统稳定性和负载能力
2. 应用端改造较小，不需要拆分业务模块

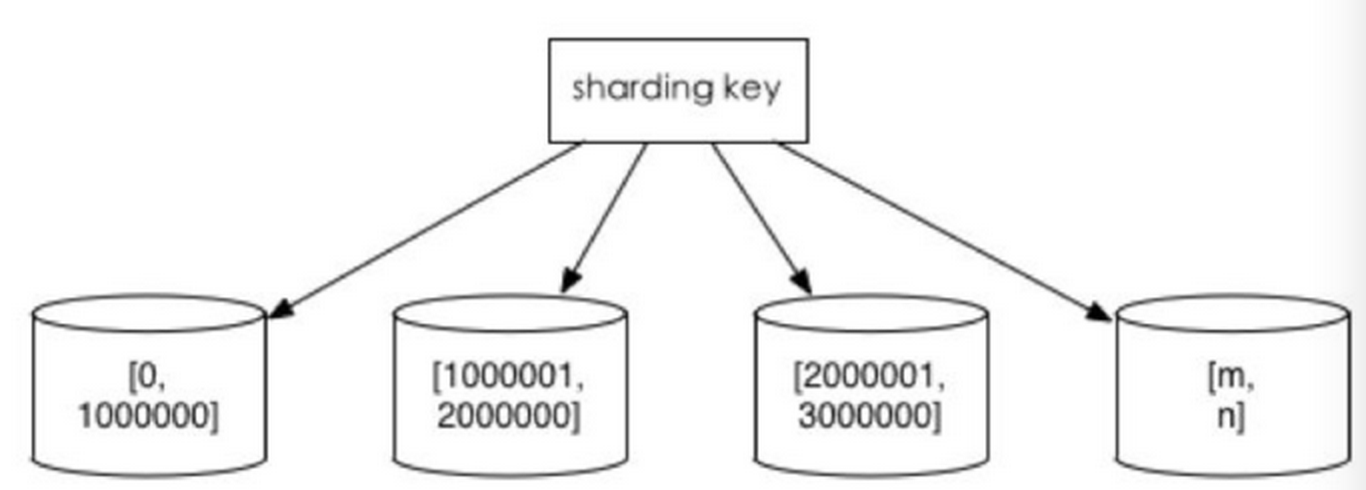
### 缺点：

1. 跨分片的事务一致性难以保证
2. 跨库的join关联查询性能较差
3. 数据多次扩展难度和维护量很大

### 水平切分的典型数据分片规则：

#### 1.根据数值范围

按照时间区间或ID区间来切分。例如：按日期将不同月甚至是日的数据分散到不同的库中；将userId为1~9999的记录分到第一个库，10000~20000的分到第二个库，以此类推。某种意义上，某些系统中使用的"冷热数据分离"，将一些使用较少的历史数据迁移到其他库中，业务功能上只提供热点数据的查询，也是类似的实践。



优点：

1. 单表大小可控
2. 天然便于水平扩展，后期如果想对整个分片集群扩容时，只需要添加结点即可，无序对其他分票的数据进行迁移。
3. 使用分片字段进行范围查找时，连续分片可以快速定位到分片进行快速查询，有效避免跨分片查询的问题。

缺点：

1. 热点数据成为瓶颈，连续分片可能存在数据热点，如有些分片上的数据读写频繁，而有些分片上的数据操作很少。

#### 2、根据数值取模

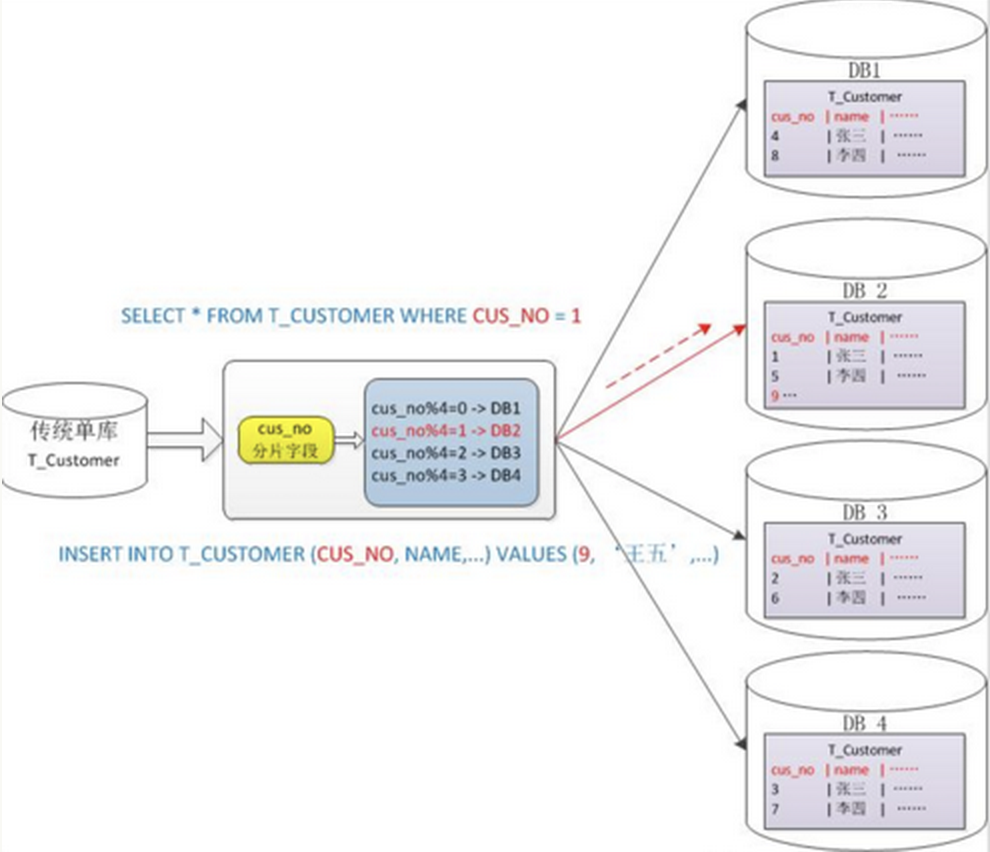
一般采用hash取模mod的切分方式，例如：将 Customer 表根据 cusno 字段切分到4个库中，余数为0的放到第一个库，余数为1的放到第二个库，以此类推。这样同一个用户的数据会分散到同一个库中，如果查询条件带有cusno字段，则可明确定位到相应库去查询。

优点：

1. 数据分片相对均匀，不容易出现热点和并发访问的瓶颈、

缺点：

1. 后期分片集群进行扩容时，需要迁移旧的数据
2. 荣i面临跨分片出巡的复杂问题。容易面临跨分片查询的复杂问题。比如上例中，如果频繁用到的查询条件中不带cusno时，将会导致无法定位数据库，从而需要同时向所有库发起查询，再在内存中合并数据，取最小集返回给运用。分库反而称为拖累。



# 二. 分库分表带来的问题

## 1、事务一致性问题

当更新内容同时分布在不同库中，不可避免会带来跨库事务问题。跨分片事务也是分布式事务，没有简单的方案，一般可使用‘XA协议’和‘两阶段提交’处理。

分布式事务能最大限度保证了数据库操作的原子性。但在提交事务时需要协调多个结点，退后了提交事务的时间点，延长了事务的执行时间，导致事务在访问共享资源时发生冲突域的概率增高，随着数据库节点的增多，这种趋势会越来越严重，从而成为系统在数据库层面上水平扩展的枷锁。

## 最终一致性

对于那些性能要求很高，但对一致性要求不高的系统，往往不苛求系统的实时一致性，只要在允许的时间段内达到最终一致性即可，可采用事务补偿的方式。与事务在执行中发生错误后立即回滚的方式不同，事务补偿是一种事后检查补救的措施，一些常见的实现方法有：对数据进行对账检查，基于日志进行对比，定期同标准数据来源进行同步等等。事务补偿还要结合业务系统来考虑。

2、跨节点关联查询 join 问题

切分之前，系统中很多列表和详情页所需的数据可以通过sql join来完成。而切分之后，数据可能分布在不同的节点上，此时join带来的问题就比较麻烦了，考虑到性能，尽量避免使用join查询。

解决这个问题的一些方法：

1）全局表

全局表，也可看做是"数据字典表"，就是系统中所有模块都可能依赖的一些表，为了避免跨库join查询，可以将这类表在每个数据库中都保存一份。这些数据通常很少会进行修改，所以也不担心一致性的问题。

2）字段冗余

一种典型的反范式设计，利用空间换时间，为了性能而避免join查询。例如：订单表保存userId时候，也将userName冗余保存一份，这样查询订单详情时就不需要再去查询"买家user表"了。

3）数据组装

在系统层面，分两次查询，第一次查询的结果集中找出关联数据id，然后根据id发起第二次请求得到关联数据。最后将获得到的数据进行字段拼装。

4）ER分片

关系型数据库中，如果可以先确定表之间的关联关系，并将那些存在关联关系的表记录存放在同一个分片上，那么就能较好的避免跨分片join问题