# 分库

## 概述

分库，是指将一个应用的数据库分拆为多个数据库。例如，一个新闻网站，原始状态，用户表、新闻表、评论表都在同一个数据库。后期，将用户表数据放到单独的用户数据库中，将评论表的数据放到单独的评论数据库中。

分表解决的是数据量多大的问题，分库解决的是数据库性能瓶颈问题。

单纯的分表可以解决数据量过大导致索引变慢的问题，但是无法解决多并发请求访问同一个库导致数据库响应变慢的问题，所以通常水平拆分都至少要采用分库的方式（垂直拆分采用分表的方式），用于一并解决大数据量和高并发的问题，这也是部分开源的分片疏忽句酷中间件支支持分库的原因。

## 垂直拆分

将系统中不存在关联关系或者需要join的表可以放在不同的数据库不同的服务器中。

按照业务垂直划分，比如可以按照业务分为资金、会员、订单三个数据库。

需要解决的问题：跨数据库的事务、join查询等问题。

## 水平拆分

按照规则划分，一般水平分库是在垂直分库之后的。比如每天处理的订单数量是海量的，可以按照一定的规则水平划分。比如大部分的站点，数据都是和用户相关，那么可以根据用户，将数据按照用户水平拆分。

需要解决的问题：数据路由、组装。

## 作用

其主要目的是为突破单节点数据库服务器I/O能力限制，解决数据库拓展性问题。

## 应用

什么时候考虑使用分库？

1. 单台DB的存储空间不够；
2. 随着查询量的增加单台数据库服务器已经没办法支撑。

但分表也有不可替代的适用场景。最常见的分表需求是事务问题，同在一个库则不需要考虑考虑分布式事务，善于使用同库不同表可有效避免分布式事务带来的麻烦。目前强一致性的分布式事务由于性能问题，导致使用起来并不一定比不分库分表快。目前采用最终一致性的柔性事务居多。分表的另一个存在的理由是，过多的数据库实力不利于运维管理。

综上所述，最佳实践是合理地配合使用分库+分表。

# 分表

## 概述

关系型数据库在大于一定数据量的情况下检索性能会急剧下降。在面对互联网海量数据情况时，所有数据都存于一张表，显然会轻易超过数据库可承受的数据量阈值。这个单表可承受的数据量阈值，需根据数据库和并发量的差异，通过实际测试获得。

分表，从表面意思上看呢，就是把一张表分成N多个小表。例如，将用户表user的数据分拆到活跃用户表 user\_active 和非活跃用户表 user\_inaction。

水平拆分如果能预估规模，越早造成本越低。

分区就是把一张表的数据分成N个区块，在逻辑上看最终只是一张表，但底层是由N个物理区块组成的。

分表就是把一张表按照一定的规则分解成N个具有独立存储空间的实体表。系统读写时需要根据定义好的规则得到对应的字表名，然后操作它。

## 作用

分表后，单表的兵法能力提高了，磁盘I/O性能也提高了，写操作效率提高了：

1. 查询一次的时间短了；
2. 数据分布在不同的文件，磁盘I/O性能提高；
3. 读写锁影响的数据量小；
4. 插入数据库需要重新建立索引的数据减少。

## 应用

什么时候考虑分表？

1. 一张表的查询速度已经慢到影响使用的时候；
2. sql经过优化；
3. 数据量大；
4. 当频繁插入或者联合查询时，速度变慢

# 分区

## 概述

分区，是利用MySQL的一种特性。数据分区是一种物理数据库的设计技术，它的目的是为了在特定的SQL操作中减少数据读写的总量以缩减响应时间。

分区就是把一张表的数据分成N多个区块，这些区块可以在同一个磁盘上，也可以在不同的磁盘上。

分区并不是生成新的数据表，而是将表的数据均衡分摊到不同的硬盘，系统或是不同服务器存储介质中，实际上还是一张表（对用户来说，逻辑上，分区后的表仍然是一个逻辑表）。另外，分区可以做到将表的数据均衡到不同的地方，提高数据检索的效率，降低数据库的频繁I/O压力值，分区的优点如下：

1. 相对于单个文件系统或是硬盘，分区可以存储更多的数据；
2. 数据管理比较方便，比如要清理或废弃某年的数据，就可以直接删除该日期的分区数据即可；
3. 精确定位分区查询数据，不需要全表扫描查询，大大提高数据检索效率；
4. 可跨多个分区磁盘查询，来提高查询的吞吐量；
5. 在涉及聚合函数查询时，可以很容易进行数据的合并。

## 水平分区

这种形式分区是对表的行进行分区，通过这样的方式不同分组里面的物理列分割的数据集得以组合，从而进行个体分割（单分区）或集体分割（1个或多个分区）。所有在表中定义的列在每个数据集中都能得到，所以表的特性依然得以保持。

## 垂直分区

这种分区方式一般来说是通过对表的垂直划分来减少目标表的宽度，使某些特定的列被划分到特定的分区，每个分区都包含了其中的列所对应的行。

## 操作

1、分区，使用MySQL自身的语法。分区的SQL语句如下：

CREATE TABLE sales (order\_date DATETIME NOT NULL)

ENGINE=InnoDB

PARTION BY RANGE(YEAR(order\_date))

(

PARTION p\_2010 VALUES LESS THAN (2010),

PARTION p\_2011 VALUES LESS THAN (2011),

PARTION p\_2012 VALUES LESS THAN (2012),

PARTION p\_catchall VALUES LESS THAN MAXVALUE

);

执行之后，在mysql/data/test目录下发现两个文件sales.frm（对象结构定义文件）和用于存储表对象的结构sales.par（应该是分区语句创造的，和分区有关）。

再执行普通的建表SQL语句：

CREATE TABLE sales2 (order\_date DATETIME NOT NULL)

ENGINE=InnoDB;

执行之后，在mysql/data/test目录下生成了文件sales2.frm

2、在information\_schema表中执行SQL语句

SELECT \* FROM PARTITIONS WHERE PARTITION\_NAME IS NOT NULL\G

看到了与上面的建分区表SQL对应的分区信息。

3、往分区表中插入数据

INSERT INTO sales (order\_date) VALUES (2007-01-01);

INSERT INTO sales (order\_date) VALUES (2008-01-01);

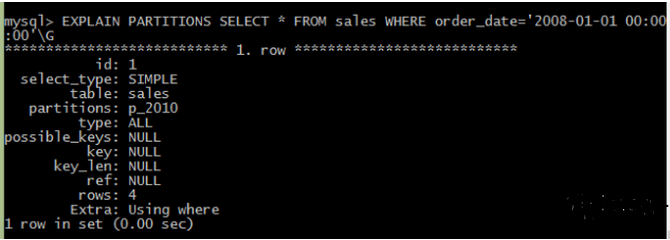
INSERT INTO sales (order\_date) VALUES (2009-01-01);

INSERT INTO sales (order\_date) VALUES (2011-01-01);

INSERT INTO sales (order\_date) VALUES (2011-01-01);

4、检测数据是否按分区存储了，执行下面的SQL语句

EXPLAIN PARTITIONS SELECT \* FROM sales WHERE order\_date = '2008-01-01 00:00:00'\G

查询结果如图：

# 对比

## 区别

### 分库+分表

分库分表区别：

1、什么是分库分表？

从字面上简单理解，分库就是把原本存储于一个库的数据分块存储到多个库上，分表就是把原本存储于一个表的数据分块存储到多个表上。

2、为什么要分库分表？

数据库中的数据量不一定是可控的，在未进行分库分表的情况下，随着时间和业务的发展，库中的表会越来越多，表中的数据量也会越来越大，相应地，数据操作，增删改查的开销也会越来越大；另外，一台服务器的资源（CPU、磁盘、内存、IO等）是有限的，最终数据库所能承载的数据量、数据处理能力都将遭遇瓶颈,。

3、分库分表的实施策略

如果你的单机性能很低了，那可以尝试分库。分库，业务透明，在物理实现上分成多个服务器，不同的分库在不同服务器上。分区可以把表分到不同的硬盘上，但不能分配到不同服务器上。一台机器的性能是有限制的，用分库可以解决单台服务器性能不够，或者成本过高问题。

当分区之后，表还是很大，处理不过来，这时候可以用分库。

orderid,userid,ordertime,.....

userid%4=0，用分库1

userid%4=1，用分库2

userid%4=2, 用分库3

userid%4=3，用分库4

上面这个就是一个简单的分库路由，根据userid选择分库，即不同的服务器

4、分库分表存在的问题

4.1 事务问题

在执行分库分表之后，由于数据存储到了不同的库上，数据库事务管理出现了困难。如果依赖数据库本身的分布式事务管理功能去执行事务，将付出高昂的性能代价；如果由应用程序去协助控制，形成程序逻辑上的事务，又会造成编程方面的负担。

4.2 跨库跨表的join问题

在执行了分库分表之后，难以避免会将原本逻辑关联性很强的数据划分到不同的表、不同的库上，这时，表的关联操作将受到限制，我们无法join位于不同分库的表，也无法join分表粒度不同的表，结果原本一次查询能够完成的业务，可能需要多次查询才能完成。

4.3 额外的数据管理负担和数据运算压力。

额外的数据管理负担，最显而易见的就是数据的定位问题和数据的增删改查的重复执行问题，这些都可以通过应用程序解决，但必然引起额外的逻辑运算，例如，对于一个记录用户成绩的用户数据表userTable，业务要求查出成绩最好的100位，在进行分表之前，只需一个order by语句就可以搞定，但是在进行分表之后，将需要n个order by语句，分别查出每一个分表的前100名用户数据，然后再对这些数据进行合并计算，才能得出结果。

### 分表+分区

1、实现方式上

a) mysql的分表是真正的分表，一张表分成很多表后，每一个小表都是完正的一张表，都对应三个文件，一个.MYD数据文件，.MYI索引文件，.frm表结构文件。

Sql代码

[root@BlackGhost test]# ls |grep user

alluser.MRG

alluser.frm

user1.MYD

user1.MYI

user1.frm

user2.MYD

user2.MYI

user2.frm

简单说明一下，上面的分表呢是利用了merge存储引擎（分表的一种），alluser是总表，下面有二个分表，user1，user2。他们二个都是独立 的表，取数据的时候，我们可以通过总表来取。这里总表是没有.MYD,.MYI这二个文件的，也就是说，总表他不是一张表，没有数据，数据都放在分表里面。我们来看看.MRG到底是什么东西

Sql代码

[root@BlackGhost test]# cat alluser.MRG |more

user1

user2

#INSERT\_METHOD=LAST

从上面我们可以看出，alluser.MRG里面就存了一些分表的关系，以及插入数据的方式。可以把总表理解成一个外壳，或者是联接池。

b) 分区不一样，一张大表进行分区后，他还是一张表，不会变成二张表，但是他存放数据的区块变多了。

[root@BlackGhost test]# ls |grep aa

aa#P#p1.MYD

aa#P#p1.MYI

aa#P#p3.MYD

aa#P#p3.MYI

aa.frm

aa.par

从上面我们可以看出，aa这张表，分为二个区，p1和p3，本来是三个区，被我删了一个区。我们都知道一张表对应三个文件.MYD,.MYI,.frm。分 区呢根据一定的规则把数据文件和索引文件进行了分割，还多出了一个.par文件，打开.par文件后你可以看出他记录了，这张表的分区信息，根分表中 的.MRG有点像。分区后，还是一张，而不是多张表。

如:

orderid,userid,ordertime,.....

ordertime<2015-01-01 #p0

ordertime<2015-04-01 #p1

ordertime<2015-07-01 #p2

ordertime<2015-10-01 #p3

ordertime<2016-01-01 #p4

按照时间分区。大部分只查询最近的订单数据，那么大部分只访问一个分区，比整个表小多了，数据库可以更加好的缓存，性能也提高了。这个是数据库分的，应用程序透明，无需修改。

2、数据处理上

a) 分表后，数据都是存放在分表里，总表只是一个外壳，存取数据发生在一个一个的分表里面。看下面的例子：

select \* from alluser where id='12'表面上看，是对表alluser进行操作的，其实不是的。是对alluser里面的分表进行了操作。

b) 分区，不存在分表的概念，分区只不过把存放数据的文件分成了许多小块，分区后的表呢，还是一张表。数据处理还是由自己来完成。

3、提高性能上

a) 分表后，单表的并发能力提高了，磁盘I/O性能也提高了。并发能力为什么提高了呢，因为查寻一次所花的时间变短了，如果出现高并发的话，总表可以根据不同的查询，将并发压力分到不同的小表里面。磁盘I/O性能怎么搞高了，本来一个非常大的.MYD文件现在也分摊到各个小表的.MYD中去了。

b) mysql提出了分区的概念，我觉得就想突破磁盘I/O瓶颈，想提高磁盘的读写能力，来增加mysql性能。

在这一点上，分区和分表的测重点不同，分表重点是存取数据时，如何提高mysql并发能力上；而分区呢，如何突破磁盘的读写能力，从而达到提高mysql性能的目的。

4)、实现的难易度上

a) 分表的方法有很多，用merge来分表，是最简单的一种方式。这种方式根分区难易度差不多，并且对程序代码来说可以做到透明的。如果是用其他分表方式就比分区麻烦了。

b) 分区实现是比较简单的，建立分区表，根建平常的表没什么区别，并且对开代码端来说是透明的。

## 联系

分表+分区

1、都能提高mysql的性高，在高并发状态下都有一个良好的表面。

2、分表和分区不矛盾，可以相互配合的，对于那些大访问量，并且表数据比较多的表，我们可以采取分表和分区结合的方式（如果merge这种分表方式，不能和分区配合的话，可以用其他的分表试），访问量不大，但是表数据很多的表，我们可以采取分区的方式等。

# 应用场景

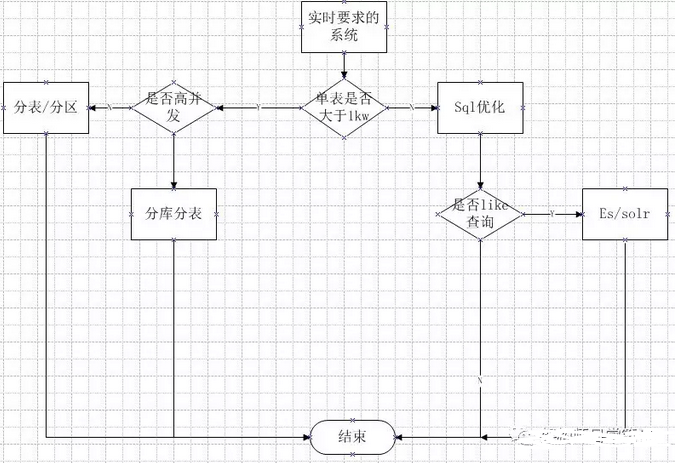
对实时性要求比较高的场景，使用数据库的分区分表分库。

对实时性要求不高的场景，可以考虑使用索引库（ElasticSearch /solr）或者大数据hadoop平台来解决（如数据分析，挖掘，报表等）或者混合使用（如ElasticSearch +hbase/mongodb）。

分区解决冷热数据分离的问题;

分库解决互联网的高并发问题；

分表解决还联网的高容量问题；

分库分表解决高并发和高容量的问题。

**实时数据**

1、1000w以下的表通常不需要分库分表，如果数据增量比较小，可以分成冷热数据，考虑分区。

2、5000w以上的表通常要考虑进行分表(不考虑并发的情况)，此时以查询为主。

3、对并发要求比较高的系统，要考虑分库，此时分表解决不了并发的要求。

4、对大并发和高容量的可以使用分库分表方案解决

**非实时数据**

1、1000w以下的读可以通过从库来读取。

2、5000w以上的大量读可通过ElasticSearch /mongodb/hbase或者混合使用。

3、数据分析，挖掘，报表等最好不要和现有系统耦合，使用单独的库或者hadoop 平台来解决。