# 表

## 复制表

## 派生表

当主查询中包含派生表，或者当select语句中包含union子句，或者select语句语句中包含一个字段的order by子句（对另一个字段的group by子句）时，MySQL为了完成查询，则需要自动创建临时表存储临时结果集，这种临时表是由MySQL自行创建，自行维护。

对于自动创建的临时表，由于内存临时表的性能更加优越，MySQL总是首先使用内存临时表，而当内存临时表变得太大时，达到某个预知的时候，内存临时表就转存为外存临时表。也就是说，外存临时表是内存临时表在存储空间上的一种延伸。内存临时表转存为外存临时表的阈值由系统变量max\_heap\_table\_size和tmp-table\_size的较小值决定。

派生表是从select语句返回的虚拟表，派生表类似于临时表，但是在select语句中使用派生表比临时表简单的多，因此它不需要创建临时表的步骤，所以当select语句的from子句中使用独立子查询时，我们将其称为派生表。

派生表一般在from子句中使用，如：

select \* from (select \* from table) table1;

## 临时表

当操作非常大的表时，你可能偶尔需要运行很多查询获得一个大量数据小的子集，不是对整个表运行这些查询，而是让MySQL每次找出所需的少数记录，将记录选择到一个临时表可能更快些，然后再这些表上运行查询。

创建临时表很容易，给正常的CREATE TABLE语句加上TEMPORARY关键字即可。

临时表将在连接MySQL期间存在，当你断开的时候，MySQL将自动删除表并释放所有空间，当然你可以在仍然连接的时候删除表并释放空间。

如果在你创建名为tmp\_table临时表时名为tmp\_table的表在数据库中已经存在，临时表将有必要屏蔽（隐藏）非临时表tmp\_table。

如果声明临时表是一个HEAP表，MySQL允许你指定在内存中创建它：

CREATE TEMPORARY TABLE tmp\_table(

)TYPE=HEAP;

因为HEAP表存储在内存中，对它运行的查询比磁盘上的临时表快些。然而，HEAP表与一般的表有些不同，且有自身的限制。

# 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。

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。

## 作用

分片解决的是拓展性问题，属于水平拆分，引入分片，就引入了数据路由和分区键的概念。

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。

## 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);

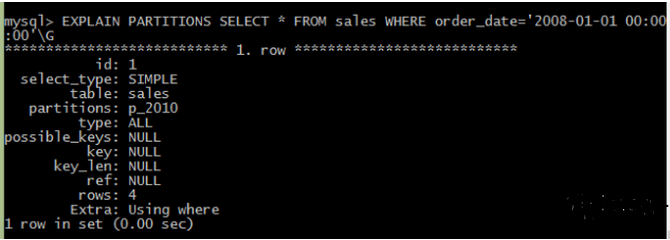
INSERT INTO sales (order\_date) VALUES (2009-01-01);

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：