**一、分区的概念**

        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子中，实际上还是一张表。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IO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、精准定位分区查询数据，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；

**二、分类 （row 行 ，column 列）**

**1、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找到，所以表的特性依然得以保持。

举个简单例子：一个包含十年发票记录的表可以被分区为十个不同的分区，每个分区包含的是其中一年的记录。（朋奕注：这里具体使用的分区方式我们后面再说，可以先说一点，一定要通过某个属性列来分割，譬如这里使用的列就是年份）

**2、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。

举个简单例子：一个包含了大text和BLOB列的表，这些text和BLOB列又不经常被访问，这时候就要把这些不经常使用的text和BLOB了划分到另一个分区，在保证它们数据相关性的同时还能提高访问速度。

在数据库供应商开始在他们的数据库引擎中建立分区（主要是水平分区）时，DBA和建模者必须设计好表的物理分区结构，不要保存冗余的数据（不同表中同时都包含父表中的数据）或相互联结成一个逻辑父对象（通常是视图）。这种做法会使水平分区的大部分功能失效，有时候也会对垂直分区产生影响。

**三、分区、分表、分库的详细理解**

**一、什么是分区、分表、分库**

**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明，然后操作它。

**分库**

一旦分表，一个库中的表会越来越多

将整个数据库比作图书馆，一张表就是一本书。当要在一本书中查找某项内容时，如果不分章节，查找的效率将会下降。而同理，在数据库中就是分区。

**二、常用的单机数据库的瓶颈**

**问题描述**

* 单个表数据量越大，读写锁，插入操作重新建立索引效率越低。
* 单个库数据量太大（一个数据库数据量到1T-2T就是极限）
* 单个数据库服务器压力过大
* 读写速度遇到瓶颈（并发量几百）

**三、分区**

**什么时候考虑使用分区？**

* 一张表的查询速度已经慢到影响使用的时候。
* sql经过优化
* 数据量大
* 表中的数据是分段的
* 对数据的操作往往只涉及一部分数据，而不是所有的数据

**分区解决的问题**

主要可以提升查询效率

**分区的实现方式（简单）**

mysql5 开始支持分区功能

1. CREATE TABLE sales (
2. id INT AUTO\_INCREMENT,
3. amount DOUBLE NOT NULL,
4. order\_day DATETIME NOT NULL,
5. PRIMARY KEY(id, order\_day)
6. ) ENGINE=Innodb
7. PARTITION BY RANGE(YEAR(order\_day)) (
8. PARTITION p\_2010 VALUES LESS THAN (2010),
9. PARTITION p\_2011 VALUES LESS THAN (2011),
10. PARTITION p\_2012 VALUES LESS THAN (2012),
11. PARTITION p\_catchall VALUES LESS THAN MAXVALUE);

**四、分表**

**什么时候考虑分表？**

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

**分表解决的问题**

分表后，单表的并发能力提高了，磁盘I/O性能也提高了，写操作效率提高了

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

**分表的实现方式（复杂）**

需要业务系统配合迁移升级，工作量较大

**分区和分表的区别与联系**

* 分区和分表的目的都是减少数据库的负担，提高表的增删改查效率。
* 分区只是一张表中的数据的存储位置发生改变，分表是将一张表分成多张表。
* 当访问量大，且表数据比较大时，两种方式可以互相配合使用。
* 当访问量不大，但表数据比较多时，可以只进行分区。

**常见分区分表的规则策略（类似）**

1. Range（范围）
2. Hash（哈希）
3. 按照时间拆分
4. Hash之后按照分表个数取模
5. 在认证库中保存数据库配置，就是建立一个DB，这个DB单独保存user\_id到DB的映射关系

**12306的订单是如何存储的？**



**五、分库**

**什么时候考虑使用分库？**

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

**分库解决的问题**

其主要目的是为突破单节点数据库服务器的 I/O 能力限制，解决数据库扩展性问题。

**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、jion查询等问题。

**水平拆分**

例如，大部分的站点。数据都是和用户有关，那么可以根据用户，将数据按照用户水平拆分。

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。需要解决的问题：数据路由、组装。

**读写分离**

对于时效性不高的数据，可以通过读写分离缓解数据库压力。需要解决的问题：在业务上区分哪些业务上是允许一定时间延迟的，以及数据同步问题。

**思路**

垂直分库-->水平分库-->读写分离

**六、拆分之后面临新的问题**

**问题**

* 事务的支持，分库分表，就变成了分布式事务
* join时跨库，跨表的问题
* 分库分表，读写分离使用了分布式，分布式为了保证强一致性，必然带来延迟，导致性能降低，系统的复杂度变高。

**常用的解决方案：**

对于不同的方式之间没有严格的界限，特点不同，侧重点不同。需要根据实际情况，结合每种方式的特点来进行处理。

选用第三方的数据库中间件（Atlas，Mycat，TDDL，DRDS），同时业务系统需要配合数据存储的升级。

**七、数据存储的演进**

**单库单表**

单库单表是最常见的数据库设计，例如，有一张用户(user)表放在数据库db中，所有的用户都可以在db库中的user表中查到。

**单库多表**

随着用户数量的增加，user表的数据量会越来越大，当数据量达到一定程度的时候对user表的查询会渐渐的变慢，从而影响整个DB的性能。如果使用mysql, 还有一个更严重的问题是，当需要添加一列的时候，mysql会锁表，期间所有的读写操作只能等待。

可以通过某种方式将user进行水平的切分，产生两个表结构完全一样的user\_0000,user\_0001等表，user\_0000 + user\_0001 + …的数据刚好是一份完整的数据。

**多库多表**

随着数据量增加也许单台DB的存储空间不够，随着查询量的增加单台数据库服务器已经没办法支撑。这个时候可以再对数据库进行水平拆分。

**八、总结**

总的来说，优先考虑分区。当分区不能满足需求时，开始考虑分表，合理的分表对效率的提升会优于分区。

**九、案例分析**

京东的商品评价存储设计，原文地址

**现状**

* 商品的评论数量：数十亿条
* 每天的服务调用：数十亿次
* 每年成倍增长

**整体的数据存储：基础数据存储，文本存储**

****

**基础数据存储**

Mysql：只存储非文本的基础信息。包括：评论状态，用户，时间等基础数据。以及图片，标签，点赞等附加信息。数据组织形式（不同的数据又可选择不同的库表拆分方案）：

* 评论基础数据按用户ID进行拆库并拆表
* 图片及标签处于同一数据库下，根据商品编号分别进行拆表
* 其它的扩展信息数据，因数据量不大、访问量不高，处理于同一库下且不做分表即可

**文本存储**

文本存储（评论的内容）使用了mongodb、hbase

* 选择nosql而非mysql
* 减轻了mysql存储压力，释放msyql，庞大的存储也有了可靠的保障
* nosql的高性能读写大大提升了系统的吞吐量并降低了延迟

转自：<http://www.cnblogs.com/bluebluesky/articles/6413831.html>

作者：[bluebluesky](http://www.cnblogs.com/bluebluesky/)

也可参考：<https://blog.csdn.net/kingcat666/article/details/78324678>

里面会有详细的说明！！我就不转载了！！

**数据分片**

**在分布式存储系统中，数据需要分散存储在多台设备上，数据分片（Sharding）就是用来确定数据在多台存储设备上分布的技术。数据分片要达到三个目的：**

1. 分布均匀，即每台设备上的数据量要尽可能相近；
2. 负载均衡，即每台设备上的请求量要尽可能相近；
3. 扩缩容时产生的数据迁移尽可能少。

**数据分片方法**

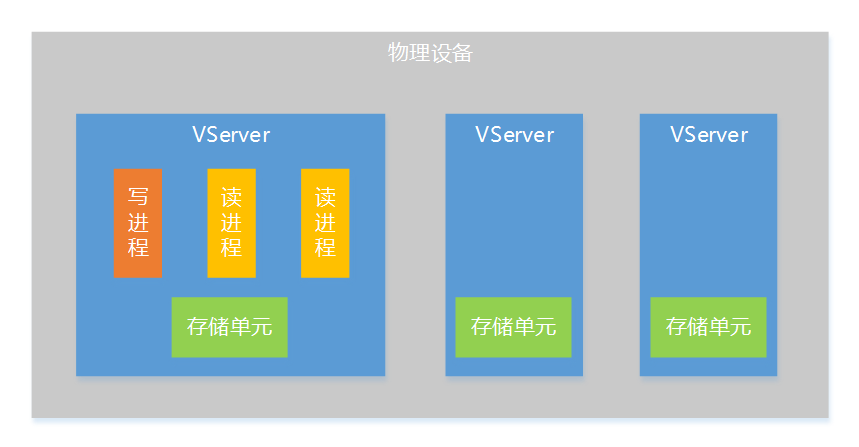
数据分片一般都是使用Key或Key的哈希值来计算Key的分布，常见的几种数据分片的方法如下：

1. 划分号段。这种一般适用于Key为整型的情况，每台设备上存放相同大小的号段区间，如把Key为[1, 10000]的数据放在第一台设备上，把Key为[10001, 20000]的数据放在第二台设备上，依次类推。这种方法实现很简单，扩容也比较方便，成倍增加设备即可，如原来有N台设备，再新增N台设备来扩容，把每台老设备上一半的数据迁移到新设备上，原来号段为[1, 10000]的设备，扩容后只保留号段[1, 5000]的数据，把号段为[5001, 10000]的数据迁移到一台新增的设备上。此方法的缺点是数据可能分布不均匀，如小号段数据量可能比大号段的数据量要大，同样的各个号段的热度也可能不一样，导致各个设备的负载不均衡；并且扩容也不够灵活，只能成倍地增加设备。
2. 取模。这种方法先计算Key的哈希值，再对设备数量取模（整型的Key也可直接用Key取模），假设有N台设备，编号为0~N-1，通过Hash(Key)%N就可以确定数据所在的设备编号。这种方法实现也非常简单，数据分布和负载也会比较均匀，可以新增任何数量的设备来扩容。主要的问题是扩容的时候，会产生大量的数据迁移，比如从N台设备扩容到N+1台，绝大部分的数据都要在设备间进行迁移。
3. 检索表。在检索表中存储Key和设备的映射关系，通过查找检索表就可以确定数据分布，这里的检索表也可以比较灵活，可以对每个Key都存储映射关系，也可结合号段划分等方法来减小检索表的容量。这样可以做到数据均匀分布、负载均衡和扩缩容数据迁移量少。缺点是需要存储检索表的空间可能比较大，并且为了保证扩缩容引起的数据迁移量比较少，确定映射关系的算法也比较复杂。
4. 一致性哈希。一致性哈希算法（Consistent Hashing）在1997年由麻省理工学院提出的一种分布式哈希（DHT）实现算法，设计目标是为了解决因特网中的热点(Hot Spot)问题，该方法的详细介绍参考此处<http://blog.csdn.net/sparkliang/article/details/5279393>。一致性哈希的算法简单而巧妙，很容易做到数据均分布，其单调性也保证了扩缩容的数据迁移是比较少的。

通过上面的对比，在这个系统选择一致性哈希的方法来进行数据分片。

**虚拟服务器**

为了让系统有更好的扩展性，这里提出存储层VServer（虚拟服务器）的概念，一个VServer是一个逻辑上的存储服务器，是分布式存储系统的一个存储单元，一台物理设备上可以部署多个VServer，一个VServer支持一个写进程和多个读进程。



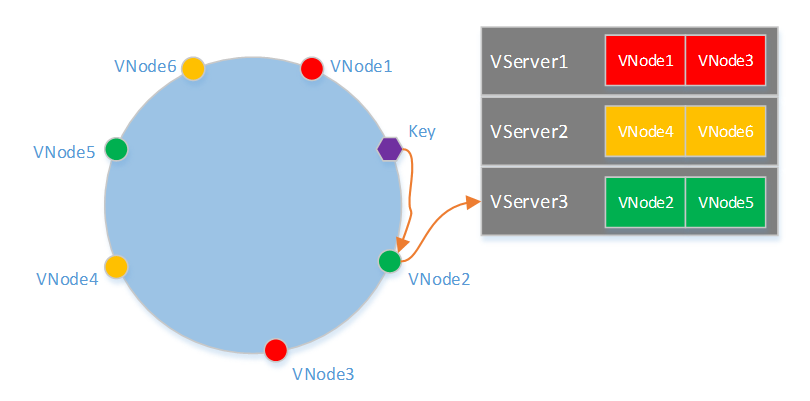
通过VServer的方式，会有下面一些好处：

1. 提高单机性能。为了不引入复杂的锁机制，采用了单写进程的设计，如果单机只有一个写进程，写并发能力会受到限制，通过VServer方式把单机上的存储资源（内存、硬盘）划分为多个存储单元，这样就支持多个写进程同时工作，大大提升单机写并发能力。
2. 部署扩展性更好。VServer的方式在部署上非常灵活，可以根据单机的资源情况来确定VServer的数量，针对不同的机型配置不同的VServer数量，这样不同的机型都能充分利用机器上的资源，即使在一个系统中使用多种机型，也能做到机器的负载比较均衡。

**一致性哈希的应用**

数据分片是在接口层实现的，目的是把数据均匀地划分到不同的VServer上。有了接口层的存在，逻辑层寻址就轻量了很多，寻址存储层VServer的工作全部由接口层负责，逻辑层只需要随机选一个接口层机器访问即可。

接口层使用了一致性哈希的割环算法来实现数据分片，在割环算法中，为了让数据均匀分布到各个VServer，每个VServer需要有多个VNode（虚拟节点）。一个Key寻址的过程如下图所示，首先根据Hash(Key)在哈希环上找到对应的VNode，在根据VNode和VServer的映射表确定所属的VServer。



由上述查找过程可知，需要事先离线计算出VNode在哈希环上的分布、VServer和VNode映射关系。为了是计算结果具有通用性，即在拥有任何数量VServer的一个系统都可以使用该结果得到一致性哈希的映射表，这就要求结果是与机器无关的，比如不能使用IP来计算VNode的哈希值。在计算前需要确定每个VServer包含的VNode数量，以及一个系统所支持的最大VServer数量。一个简单的方法是类似上文链接中提到的方法，但不能和IP相关，可以改用VServer和VNode的编号来计算哈希值，如Hash("1#1")，Hash("1#2")… 这种方法要求一个VServer包含的VNode的数量比较多，大概需要500个才能使各个VServer上的数据比较均匀。当然还有其他的一些方法做到一个VServer上包含更少的VNode数量，并且让数据分布偏差在一定范围内。

Google提出了一种新的一致性哈希算法Jump Consistent Hash，此算法零内存消耗，均匀分配，快速，并且只有5行代码，优势非常明显，详细介绍见此处<http://my.oschina.net/u/658658/blog/424161>。和上面介绍的方法相比，一个最大的不同点是，在扩容重新分布数据时，在上面的方法中，新机器的一个VNode上的数据只会来自一个老机器上的VNode，而这种方法是会来自所有老机器上的VNode。这个问题可能会导致一些设计上复杂化，所以使用的时候要慎重考虑。

转：<http://www.cnblogs.com/Leo_wl/p/5654789.html>

**分片模式是什么？**

数据的切分（Sharding）根据其切分规则的类型，可以分为两种切分模式。

（1）一种是按照不同的表（或者Schema）来切分到不同的数据库（主机）之上，这种切分可以称之为数据的垂直（纵向）切分

（2）另外一种则是根据表中的数据的逻辑关系，将同一个表中的数据按照某种条件拆分到多台数据库（主机）上面，这种切分称之为数据的水平（横向）切分。

**分片相关的概念**

逻辑库(schema) ：

通常对实际应用来说，并不需要知道中间件的存在，业务开发人员只需要知道数据库的概念，所以数据库中间件可以被看做是一个或多个数据库集群构成的逻辑库。

* 逻辑表（table）：

既然有逻辑库，那么就会有逻辑表，分布式数据库中，对应用来说，读写数据的表就是逻辑表。逻辑表，可以是数据切分后，分布在一个或多个分片库中，也可以不做数据切分，不分片，只有一个表构成。

* 分片表：

是指那些原有的很大数据的表，需要切分到多个数据库的表，这样，每个分片都有一部分数据，所有分片构成了完整的数据。 总而言之就是需要进行分片的表。

* 非分片表：

一个数据库中并不是所有的表都很大，某些表是可以不用进行切分的，非分片是相对分片表来说的，就是那些不需要进行数据切分的表。

* 分片节点(dataNode)

数据切分后，一个大表被分到不同的分片数据库上面，每个表分片所在的数据库就是分片节点（dataNode）。

* 节点主机(dataHost)

数据切分后，每个分片节点（dataNode）不一定都会独占一台机器，同一机器上面可以有多个分片数据库，这样一个或多个分片节点（dataNode）所在的机器就是节点主机（dataHost）,为了规避单节点主机并发数限制，尽量将读写压力高的分片节点（dataNode）均衡的放在不同的节点主机（dataHost）。

* 分片规则(rule)

前面讲了数据切分，一个大表被分成若干个分片表，就需要一定的规则，这样按照某种业务规则把数据分到某个分片的规则就是分片规则，数据切分选择合适的分片规则非常重要，将极大的避免后续数据处理的难度。

**一: 分区简介  
分区是根据一定的规则，数据库把一个表分解成多个更小的、更容易管理的部分。就访问数据库应用而言，逻辑上就只有一个表或者一个索引，但实际上这个表可能有N个物理分区对象组成，每个分区都是一个独立的对象，可以独立处理，可以作为表的一部分进行处理。分区对应用来说是完全透明的，不影响应用的业务逻辑。**

分区有利于管理非常大的表，它采用分而治之的逻辑，分区引入了分区键的概念，分区键用于根据某个区间值(或者范围值)、特定值列表或者hash函数值执行数据的聚集，让数据根据规则分布在不同的分区中，让一个大对象碧昂城一些小对象。

MySQL分区即可以对数据进行分区也可以对索引进行分区。

分区类型  
range分区：基于一个给定的连续区间范围(区间要求连续并且不能重叠)，把数据分配到不同的分区  
list分区：类似于range分区，区别在于list分区是居于枚举出的值列表分区，range是基于给定的连续区间范围分区  
hash分区：基于给定的分区个数，把数据分配到不同的分区  
key分区：类似于hash分区  
注意：无论哪种分区，要么你分区表上没有主键/唯一键，要么分区表的主键/唯一键都必须包含分区键，也就是说不能使用主键/唯一键字段之外的其它字段分区。

####MySQL分区的有限主要包括以下4个方面：

和单个磁盘或者文件系统分区相比，可以存储更多数据  
优化查询。在where子句中包含分区条件时，可以只扫描必要的一个或者多个分区来提高查询效率；同时在涉及sum()和count()这类聚合函数的查询时，可以容易的在每个分区上并行处理，最终只需要汇总所有分区得到的结果  
对于已经过期或者不需要保存的数据，可以通过删除与这些数据有关的分区来快速删除数据  
跨多个磁盘来分散数据查询，以获得更大的查询吞吐量  
分区和水平分表功能类似，将一个大表的数据分割到多张小表中去，由于查询不需要全表扫描了，只需要扫描某些分区，所以分区能提高查询速度。

水平分表需要用户预先手动显式创建出多张分表(如tbl\_user0, tbl\_user1, tbl\_user2)，在物理上实实在在的创建多张表，通过客户端代理(Sharding-JDBC等)或者中间件代理(Mycat等)来实现分表逻辑。

分区是MySQL的一个插件Plugin功能，将一张大表的数据在数据库底层分成多个分区文件(如tbl\_user#P#p0.ibd, tbl\_user#P#p1.ibd, tbl\_user#P#p2.ibd)，和水平分表不同的是分区不需要显式的创建“分表”,数据库会自动创建分区文件的，用户看到的只是一张普通的表，其实是对应的是多个分区，这个是对用户是屏蔽的、透明的，在使用上和使用一张表完全一样，不需要借助任何功能来实现。分区是一种逻辑上的水平分表，在物理层面还是一张表。

二：数据库文件  
CREATE TABLE `tbl\_user\_innodb` (  
   `id` int(11) NOT NULL AUTO\_INCREMENT,  
   `username` varchar(255) DEFAULT NULL,  
   `email` varchar(20) DEFAULT NULL,  
   `age` tinyint(4) DEFAULT NULL,  
   `type` int(11) DEFAULT NULL,  
   `create\_time` datetime DEFAULT CURRENT\_TIMESTAMP,  
   PRIMARY KEY (`id`)  
 ) ENGINE=InnoDB AUTO\_INCREMENT=5100002 DEFAULT CHARSET=utf8;  
   
 CREATE TABLE `tbl\_user\_myisam` (  
   `id` int(11) NOT NULL AUTO\_INCREMENT,  
   `username` varchar(255) DEFAULT NULL,  
   `email` varchar(20) DEFAULT NULL,  
   `age` tinyint(4) DEFAULT NULL,  
   `type` int(11) DEFAULT NULL,  
   `create\_time` datetime DEFAULT CURRENT\_TIMESTAMP,  
   PRIMARY KEY (`id`)  
 ) ENGINE=myisam AUTO\_INCREMENT=5100002 DEFAULT CHARSET=utf8;  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
通过show variables like ‘%datadir%’;命令查看mysql的data存放目录，进入所在的数据库目录(如test)，不同的引擎数据库文件格式不同

myisam  
.frm : 存储表结构  
.MYD    : 存储表数据  
.MYI    : 存储索引文件  
innodb: 只有设置成独立表空间才能做成功表分区  
.frm : 表结构  
.ibd : 数据 + 索引

三：插入500W条数据  
CREATE TABLE `tbl\_user\_no\_part` (  
   `id` int(11) NOT NULL AUTO\_INCREMENT,  
   `username` varchar(255) DEFAULT NULL,  
   `email` varchar(20) DEFAULT NULL,  
   `age` tinyint(4) DEFAULT NULL,  
   `type` int(11) DEFAULT NULL,  
   `create\_time` datetime DEFAULT CURRENT\_TIMESTAMP,  
   PRIMARY KEY (`id`)  
) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;

-- 修改mysql默认的结束符号，默认是分号；但是在函数和存储过程中会使用到分号导致解析不正确  
delimiter $$

-- 随机生成一个指定长度的字符串  
create function rand\_string(n int) returns varchar(255)   
begin   
 # 定义三个变量  
 declare chars\_str varchar(100) default 'abcdefghijklmnopqrstuvwxyzABCDEFJHIJKLMNOPQRSTUVWXYZ';  
 declare return\_str varchar(255) default '';  
 declare i int default 0;

 while i < n do   
   set return\_str = concat(return\_str, substring(chars\_str, floor(1+rand()\*52), 1));  
   set i = i + 1;  
 end while;  
 return return\_str;  
end $$

-- 创建插入的存储过程  
create procedure insert\_user(in start int(10), in max\_num int(10))  
begin  
    declare i int default 0;   
    set autocommit = 0;    
    repeat  
        set i = i + 1;  
        insert into tbl\_user\_no\_part values ((start+i) ,rand\_string(8), concat(rand\_string(6), '@random.com'), 1+FLOOR(RAND()\*100), 3, now());  
        until i = max\_num  
    end repeat;  
   commit;  
end $$

-- 将命令结束符修改回来  
delimiter ;

-- 调用存储过程，插入500万数据，需要等待一会时间，等待执行完成  
call insert\_user(100001,5000000);  
-- Query OK, 0 rows affected (7 min 49.89 sec) 我的Macbook Pro i5 8G内存用了8分钟才执行完

select count(\*) from tbl\_user\_no\_part;  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
四：range分区  
MySQL有五种分区类型 range、list、hash、key、子分区，其中最常用的是range和list分区

-- 查看mysql版本  
select version();

-- 查看分区插件是否激活 partition active  
show plugins;

对于低版本的MySQL，如果InnoDB引擎要想分区成功，需要在my.conf中设置innodb\_file\_per\_table=1 设置成独立表空间  
独立表空间：每张表都有对应的.ibd文件  
innodb\_file\_per\_table=1  
1  
2  
3  
4  
5  
6  
7  
8  
9  
range分区：给定一个连续区间的范围值进行分区，某个字段的值满足这个范围就会被分配到该分区。适用于字段的值是连续的区间的字段，如 日期范围, 连续的数字

-- 语法  
create table <table> (  
    // 字段  
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 AUTO\_INCREMENT=1  
partition by range (分区字段) (  
  partition <分区名称> values less than (Value),  
  partition <分区名称> values less than (Value),  
  ...  
  partition <分区名称> values less than maxvalue  
);  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
range：表示按范围分区  
分区字段：表示要按照哪个字段进行分区，可以是一个字段名，也可以是对某个字段进行表达式运算如year(create\_time)，使用range最终的值必须是数字  
分区名称: 要保证不同，也可以采用 p0、p1、p2 这样的分区名称，  
less than : 表示小于  
Value : 表示要小于某个具体的值，如 less than (10) 那么分区字段的值小于10的都会被分到这个分区  
maxvalue: 表示一个最大的值

注意：range 对应的分区键值必须是数字值，可以使用range columns(分区字段) 对非int型做分区，如字符串，对于日期类型的可以使用year()、to\_days()、to\_seconds()等函数

create table emp\_date(  
    id int not null,  
    separated date not null default '9999-12-31'  
)  
partition by range columns(separated) (  
    partiontion p0 values less than ('1990-01-01'),  
    partiontion p0 values less than ('2001-01-01'),  
    partiontion p0 values less than ('2018-01-01')  
);  
1  
2  
3  
4  
5  
6  
7  
8  
9  
分区可以在创建表的时候进行分区，也可以在创建表之后进行分区

alter table <table> partition by RANGE(id) (  
    PARTITION p0 VALUES LESS THAN (1000000),  
    PARTITION p1 VALUES LESS THAN (2000000),  
    PARTITION p2 VALUES LESS THAN (3000000),  
    PARTITION p3 VALUES LESS THAN (4000000),  
    PARTITION p4 VALUES LESS THAN MAXVALUE   
);  
1  
2  
3  
4  
5  
6  
7  
-- 创建分区表  
CREATE TABLE `tbl\_user\_part` (  
   `id` int(11) NOT NULL ,  
   `username` varchar(255) DEFAULT NULL,  
   `email` varchar(20)     DEFAULT NULL,  
   `age` tinyint(4)        DEFAULT NULL,  
   `type` int(11)          DEFAULT NULL,  
   `create\_time` datetime  DEFAULT CURRENT\_TIMESTAMP  
   -- PRIMARY KEY (`id`,`age`)  
) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8  
PARTITION BY RANGE (age) (  
    PARTITION p0 VALUES LESS THAN (20),  
    PARTITION p1 VALUES LESS THAN (40),  
    PARTITION p2 VALUES LESS THAN (60),  
    PARTITION p3 VALUES LESS THAN (80),  
    PARTITION p4 VALUES LESS THAN MAXVALUE  
);

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18

在创建分区的时候经常会遇到这个错误：A PRIMARY KEY must include all columns in the table’s partitioning function。意思是说分区的字段必须是要包含在主键当中。 可以使用PRIMARY KEY (id,xxx)来将多个字段作为主键。在做分区表时，选择分区的依据字段时要谨慎，需要仔细斟酌这个字段拿来做为分区依据是否合适，这个字段加入到主键中做为复合主键是否适合。

使用range分区时表结构要么没有主键，要么分区字段必须是主键。

-- 将tbl\_user\_no\_part表中的数据复制到tbl\_user\_part表中(数据量比较多，可能要等几分钟)  
INSERT INTO tbl\_user\_part SELECT \* FROM tbl\_user\_no\_part;

SELECT count(\*) FROM tbl\_user\_no\_part WHERE age > 25 AND age < 30;  
SELECT count(\*) FROM tbl\_user\_part WHERE age > 25 AND age < 30;  
1  
2  
3  
4  
5

从查询结果看，当查询条件中包括分区字段时，分区确实能提高查询效率

五：list 分区  
设置若干个固定值进行分区，如果某个字段的值在这个设置的值列表中就会被分配到该分区。适用于字段的值区分度不高的，或者值是有限的，特别是像枚举这样特点的列。list分区使用in表示一些固定的值的列表

-- 语法  
create table <table> (  
    // 字段  
) ENGINE=数据库引擎  DEFAULT CHARSET=utf8 AUTO\_INCREMENT=1  
partition by LIST (分区字段或者基于该字段的返回的整数值的表达式) (  
  partition <分区名称> values IN (Value1，Value2, Value3),  
  ...  
  partition <分区名称> values IN (Value4, Value5),  
);  
1  
2  
3  
4  
5  
6  
7  
8  
9  
columns分区  
在mysql5.5之前range分区和list分区只支持整数分区，可以通过额外的函数运算或者额外的转换从而得到一个整数。columns分区分为 range columns 和 list columns 两种，支持整数(tinyint到bigint， 不支持decimal 和float)、日期(date、datetime)、字符串(char、varchar、binary、varbinary)三大数据类型。

columns分区支持一个或者多个字段作为分区键，不支持表达式作为分区键，这点区别于range 和 list 分区。需要注意的是range columns 分区键的比较是基于元组的比较，也就是基于字段组的比较，这和range分区有差异。

create talbe rc3 (  
    a int,  
    b int  
)  
partition by range columns(a, b) (  
    partition p01 values less than (0, 10),  
    partition p02 values less than (10, 10),  
    partition p03 values less than (10, 20),  
    partition p04 values less than (10, 35),  
    partition p05 values less than (10, maxvalue),  
    partition p06 values less than (maxvalue, maxvalue),  
);

insert into rc3(a, b) values(1, 10);

select (1, 10) < (10, 10) from dual;

-- 根据结果存放到p02分区上了  
select  
    partition\_name,  
    partition\_expression,  
    partition\_description,  
    table\_rows  
from information\_schema.partitions  
where table\_schema = schema() and table\_name = 'rc3';

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
range columns分区键的比较(元组的比较)其实就是多列排序，先根据a字段排序再根据b字段排序，根据排序结果来分区存放数据，和range单字段的分区排序的规则实际上是一样的

六：hash分区  
Hash分区主要用来分散热点读，确保数据在预先确定个数的分区中可能的平均分布。对一个表执行Hash分区时，mysql会对分区键应用一个散列函数，以此确定数据应当放在N个分区中的哪个分区。

mysql支持两种hash分区，

常规hash分区和线性hash分区(linear hash分区)，常规hash分区使用的是取模算法，对应一个表达式expr是可以计算出它被保存到哪个分区中，N = MOD(expr, num)  
线性hash分区使用的是一个线性的2的幂运算法则。  
对指定的字段(整型字段)进行哈希，将记录平均的分配到分区中，使得所有分区的数据比较平均。 hash分区只需要指定要分区的字段和要分成几个分区,  
expr是一个字段值或者基于某列值云散返回的一个整数，expr可以是mysql中有效的任何函数或者其它表达式，只要它们返回一个即非常熟也非随机数的整数。  
num 表示分区数量

-- HASH  
create table <table> (  
    // 字段  
) ENGINE=数据库引擎  DEFAULT CHARSET=utf8 AUTO\_INCREMENT=1  
PARTITION BY HASH(expr)  
PARTITIONS <num>;  
1  
2  
3  
4  
5  
6  
常规hash分区方式看上去挺不错的，通过取模的方式来数据尽可能平均分布在每个分区，让每个分区管理的数据都减少，提高查询效率，可是当我们要增加分区时或者合并分区，问题就来了，假设原来是5个常规hash分区，现在需要增加一个常规分区，原来的取模算法是MOD(expr, 5), 根据余数0~4分布在5个分区中，现在新增一个分区后，取模算法变成MOD(expr, 6),根据余数0~6分区在6个分区中，原来5个分区的数据大部分都需要通过重新计算进行重新分区。

常规hash分区在管理上带来了的代价太大，不适合需要灵活变动分区的需求。为了降低分区管理上的代价，mysql提供了线性hash分区，分区函数是一个线性的2的幂的运算法则。同样线性hash分区的记录被存在那个分区也是能被计算出来的。线性hash分区的优点是在分区维护(增加、删除、合并、拆分分区)时，mysql能够处理的更加迅速，缺点是：对比常规hash分区，线性hash各个分区之间数据的分布不太均衡。

-- LINEAR HASH  
create table <table> (  
    // 字段  
) ENGINE=数据库引擎  DEFAULT CHARSET=utf8 AUTO\_INCREMENT=1  
PARTITION BY LINEAR HASH(expr)  
PARTITIONS <num>;  
1  
2  
3  
4  
5  
6  
七：key分区  
按照key进行分区非常类似于按照hash进行分区，只不过hash分区允许使用用户自定义的表达式，而key分区不允许使用用于自定义的表达式，需要使用mysql服务器提供的hash函数，同时hash分区只支持整数分区，而key分区支持使用出blob or text类型外的其他类型的列作为分区键。

和hash功能一样，不同的是分区的字段可以是非int类型,如字符串、日期等类型。

可以使用partition by key(expr)子句来创建一个key分区表，expr是零个或者多个字段名的列表。key分区也支持线性分区linear key

partition by key(expr) partitions num;

-- 不指定默认首选主键作为分区键，在没有主键的情况下会选择非空唯一键作为分区键  
partition by key() partitions num;

-- linear key  
partition by linear key(expr)  
1  
2  
3  
4  
5  
6  
7  
8  
create table <table> (  
    // 字段  
) ENGINE=数据库引擎  DEFAULT CHARSET=utf8 AUTO\_INCREMENT=1  
PARTITION BY HASH(分区字段名)  
PARTITIONS <count>;  
1  
2  
3  
4  
5  
八：子分区  
子分区(subpartition)：是分区表中对每个分区的再次分割，又被称为复合分区，支持对range和list进行子分区，子分区即可以使用hash分区也可以使用key分区。复合分区适用于保存非常大量的数据记录。

-- 根据年进行分区  
-- 再根据天数分区  
-- 3个range分区(p0,p1,p2)又被进一步分成2个子分区，实际上整个分区被分成了 3 x 2 = 6个分区  
create table ts (  
    id int,   
    purchased date  
)   
partition by range(year(purchased))  
subpartition by hash(to\_days(purchased)) subpartitions 2   
(  
    partition p0 values less than (1990),  
    partition p0 values less than (2000),  
    partition p0 values less than maxvalue  
);  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
CREATE TABLE IF NOT EXISTS `sub\_part` (  
  `news\_id` int(11) NOT NULL  COMMENT '新闻ID',  
  `content` varchar(1000) NOT NULL DEFAULT '' COMMENT '新闻内容',  
  `u\_id`  int(11) NOT NULL DEFAULT 0s COMMENT '来源IP',  
  `create\_time` DATE NOT NULL DEFAULT '0000-00-00 00:00:00' COMMENT '时间'  
) ENGINE=INNODB  DEFAULT CHARSET=utf8  
PARTITION BY RANGE(YEAR(create\_time))  
SUBPARTITION BY HASH(TO\_DAYS(create\_time))  
(  
PARTITION p0 VALUES LESS THAN (1990) (SUBPARTITION s0, SUBPARTITION s1, SUBPARTITION s2),  
PARTITION p1 VALUES LESS THAN (2000) (SUBPARTITION s3, SUBPARTITION s4, SUBPARTITION good),  
PARTITION p2 VALUES LESS THAN MAXVALUE (SUBPARTITION tank0, SUBPARTITION tank1, SUBPARTITION tank3)  
);  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
九：管理分区  
mysql不禁止在分区键值上使用null,分区键可能是一个字段或者一个用户定义的表达式，一般情况下，mysql的分区把null值当做零值或者一个最小值进行处理。range分区中，null值会被当做最小值来处理；list分区中null值必须出现在枚举列表中，否则不被接受；hash/key分区中,null值会被当做领值来处理。

mysql提供了添加、删除、重定义、合并、拆分分区的命令，这些操作都可以通过alter table 命令来实现

-- 删除list或者range分区(同时删除分区对应的数据)  
alter table <table> drop partition <分区名称>;

-- 新增分区  
-- range添加新分区  
alter table <table> add partition(partition p4 values less than MAXVALUE);

-- list添加新分区  
alter table <table> add partition(partition p4 values in (25,26,28));

-- hash重新分区  
alter table <table> add partition partitions 4;

-- key重新分区  
alter table <table> add partition partitions 4;

-- 子分区添加新分区，虽然我没有指定子分区，但是系统会给子分区命名的  
alter table <table> add partition(partition p3 values less than MAXVALUE);

-- range重新分区  
ALTER TABLE user REORGANIZE PARTITION p0,p1,p2,p3,p4 INTO (PARTITION p0 VALUES LESS THAN MAXVALUE);

-- list重新分区  
ALTER TABLE <table> REORGANIZE PARTITION p0,p1,p2,p3,p4 INTO (PARTITION p0 VALUES in (1,2,3,4,5));  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
分区优点

1，分区可以分在多个磁盘，存储更大一点

2，根据查找条件，也就是where后面的条件，查找只查找相应的分区不用全部查找了

3，进行大数据搜索时可以进行并行处理。

4，跨多个磁盘来分散数据查询，来获得更大的查询吞吐量