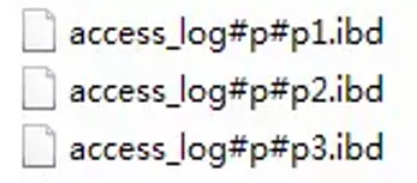
# 概述

分区是将一个表的数据按照某种方式，比如按照月、天或者其他方式，分成多个较小的、更容易管理的部分，也就是物理存储根据一定规则放在不同文件中，但是逻辑上所有的数据仍在一个表中（只是逻辑上是一个表，物理上是多个表了，有时候银行做分区是为了后续做数据按照分区迁移方便）。如下图所示：



**MySQL实现分区表的方式是对底层表的封装，意味着索引也是按照分区的子表定义的，没有全局索引（因此分布式数据库中引入全局索引就非常有必要了）。**这和Oracle不同，在Oracle中可以更加灵活的定义索引和表是否进行分区。

**分区功能并不是存储引擎层完成的**。因此不只有InnoDB存储引擎支持分区，常见的存储引擎MyISAM、NDB等都支持分区。但是并不是所有的存储引擎都支持，如CSV、FEDORATED、MERGE等就不支持分区。分区的过程是将一个表或者索引分解为多个更小、更可管理的部分。就访问数据库的应用而言，从逻辑上讲，只有一个表或者索引，但是在物理上这个表或者索引可能由十个物理分区组成。

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（这里有个前提就是条件查询中必须包含分区键，否则扫描全部分区反而很慢）**。

## 子分区

子分区（subpartitioning）是在分区的基础上再进行分区，有时也称这种分区为**复合分区（composite partitioning）**。

子分区的建立需要注意以下几个问题：

每个子分区的数量必须相同。

如果在一个分区表的任何分区上使用SUBPARTITION来明确定义任何子分区，那么就必须定义所有的子分区。

每个SUBPARTITION子句必须包括子分区的一个名字。

子分区的名字必须是唯一的。

## 特点

### 优点

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；

通过TRUNCATE操作可以快速清理特定分区数据，通过DROP操作也可以直接删除掉不需要的分区和对应的数据。

1. DML操作加锁仅影响操作的分区，不会影响未访问分区；
2. 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；
3. 通过大数据量分区能有效降低索引层数，提高查询性能；
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。

### 缺点

分区利用不好会存在性能问题。

大多数DBA会有这样一个误区：只要启用了分区，数据库就会运行得更快。这种想法是存在很多问题的。分区可能会提高某些SQL语句性能，但是其主要用于高可用性，利于数据库的管理。在OLTP应用中，对分区的使用应该非常小心。总之，如果只是一味地使用分区，而不理解分区是如何工作的，也不清楚如何使用分区，那么分区极有可能只会对性能产生负面的影响。

**具体缺点如下：**

1. DDL操作需要锁定所有分区，导致所有分区上操作都被阻塞。
2. 当表数据量较小时，分区表和非分区表性能相近，分区表效果有限。
3. 当表数据量较大时，对分区表进行DDL或其他运维操作难度大、风险高。
4. 分区表在行业内使用较少，社区资料有限，存在未知风险多。
5. 当单台服务器性能无法满足时，对分区表进行分拆的成本较高。
6. 当分区表操作不当导致访问所有分区时，会导致严重的性能问题。

7、使用分库分表可以有效降低运维操作影响，对1亿数据量表做DDL操作需要谨慎评估，而对10万数据量表做DDL操作可以默认其很快完成。

8、使用分库分表可以有效减小宕机或其他故障影响，将数据分库分表到10套群集上，一套群集发生故障仅影响一部分的业务。

9、MySQL不支持自动分区扩展，需要手动新增分区并进行数据再均衡。

## 限制

无论采用哪种分区，要么分区表上没有主键/唯一键，要么分区表的主键/唯一键都必须包含分区键，即不能使用主键/唯一键字段之外的其它字段分区。

在唯一键中不包括分区表达式中使用的所有列。因此在创建的时候就会报错：ERROR 1491 (HY000): A PRIMARY KEY must include all columns in thetable's partitioning function。

秉承一个原则：**主键/唯一键必须包含分区键的所有列**。

# 原理

分区表由多个相关的底层表实现，这些底层表也是由**句柄对象**表示，因此也可以直接访问各个分区。存储引擎管理分区的各个底层表和管理普通表一样，所有的底层表都必须使用相同的存储引擎，分区表的索引只是在各个底层表各自加上一个完全相同的索引。从存储引擎的角度来看，底层表和一个普通表没有任何不同，存储引擎也无需知道这是一个普通表还是一个分区表的一部分。

那么在分区表上的操作是怎样进行的呢？其实常规的CRUD操作以及返回结果和普通表没有任何区别。具体分区层实现是先打开并锁定所有底层表，优化器先判断是否可以过滤部分分区，然后调用对应的存储引擎接口访问各个分区的数据进行相应的操作。

问题1：为什么Oracle分区表用的很常见，MySQL却不推荐？

因为是两种不同的数据库，拿MySQL当Oracle用，会有很多不如意的地方。Oracle单库过T很正常，TP+AP很强，原生的HTAP的支持，MySQL的AP相对要弱很多，单库过T是不建议，我们的容量规划目前是按照300G的容量规格设计的，基本上从设计层面能够做到冷热数据分离和规避数据过度增长。

# 分类

MySQL数据库支持的分区类型为水平分区，并不支持垂直分区。

**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。

注：MySQL是不支持这种垂直分区的。

此外，MySQL数据库的分区是局部分区索引，一个分区中既存放数据又存放索引。全局分区是指，数据存放各个分区中，但是所有数据的索引放在一个对象中。目前，MySQL数据库暂时不支持全局分区。

# 类型

## RANGE分区

行数据基于属于一个给定连续区间的列值放入分区。MySQL 5.5开始支持RANGE COLUMNS的分区。

## LIST分区

和RANGE分区类型一样，只是LIST分区面向的是离散的值。MySQL 5.5开始支持LIST COLUMNS的分区。

## HASH分区

根据用户自定义表达式的返回值来进行分区，返回值不能为负数。

**注意：**即使是根据自增长主键进行的HASH分区也不能保证分区数据是均匀的，因为插入的自增长ID并非总是连续的，如果该主键值因为某种原因被回滚了，则该值不会再次被自动使用。

常规HASH和线性HASH增加和收缩分区原理是一样的。增加和收缩分区后原来的数据会根据现有的分区数量重新分布。HASH分区不能删除分区，所以不能使用DROP PARTITION操作进行分区删除操作。可以通过ALTER TABLE ... COALESCE PARTITION num合并分区，这里的num是减去的分区数量。可以通过ALTERTABLE ... ADD PARTITION PARTITIONS num来增加分区，这里是num是增加的分区数量。

## KEY分区

根据MySQL数据库提供的散列函数来进行分区。

和HASH分区类似，但是KEY分区不允许使用自定义的表达式，需要使用MySQL Server提供的HASH函数。

MYSQL支持两种KEY分区, 常规KEY(KEY)分区和线性KEY(LINEARKEY) 分区,其中计算区别和HASH分区一样。

注：**不论创建何种类型的分区，如果表中存在主键或唯一索引时，分区列必须是唯一索引的一个组成部分**。

**唯一索引可以是NULL值，并且只要求分区列是唯一索引的一个组成部分，不需要整个唯一索引列都是分区列**。

**当建表时没有指定主键和唯一索引时，可以指定任何一个列为分区列**。

## COLUMNS分区

在前面介绍的RANGE、LIST、HASH和KEY这四种分区中，分区的条件必须是整型（integer），如果不是整型，那么需要通过函数将其转化为整型，如YEAR()、TO\_DAYS()、MONTH()等函数。MySQL 5.5版本开始支持COLUMNS分区，可视为对RANGE分区和LIST分区的一种进化。**COLUMNS分区可以直接使用非整型的数据进行分区，分区根据类型直接比较而得到，不需要转化为整型。此外，RANGE COLUMNS分区可以对多个列的值进行分区**。

COLUMNS分区支持以下数据类型：

所有的整型类型，如INT、SMALLINT、TINYINT和BIGINT。对FLOAT和DECIMAL则不予支持。

日期类型，如DATE和DATETIME。对其余的日期类型不予支持。

字符串类型，如CHAR、VARCHAR、BINARY和VARBINARY。对BLOB和TEXT类型不予支持。对于日期类型的分区，我们不再需要YEAR()和TO\_DAYS()函数，可以直接使用COLUMNS。

# 操作

## 建立分区

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);

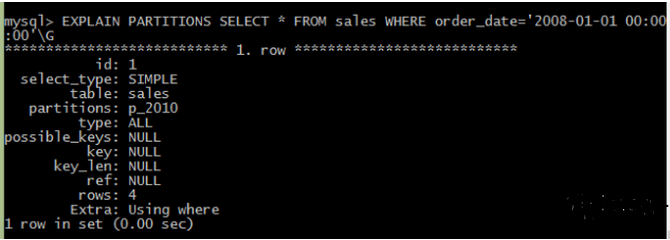
INSERT INTO sales (order\_date) VALUES (2011-01-01);

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：



## 修改分区

alter table t reorganize partition s0 into (partition s1 values in(1,2));

alter table t rename partition s0 to s1;

## 交换分区

在MySQL中，分区是可以交换的，可以将一个分区表中的一个分区和一个普通表中的数据交换。

语法：

ALTER TABLE pt EXCHANGE PARTITION p WITH TABLE nt;

说明：将分区表pt的分区p和一个普通表nt中的数据进行互换。

交换的前提条件：

1. 普通表nt不为临时表，且不是分区表
2. 表结构和分区表pt的结构一致
3. 普通表nt没有外键引用
4. 普通表nt若为非空。在MySQL5.6以及之前的版本，必须在分区的范围内；在MySQL5.7之后则可以不在其分区范围内，也依然会成功存入该分区中。

# 性能

## 分区中NULL值

MySQL数据库允许对NULL值做分区，但是处理的方法与其他数据库可能完全不同。MySQL数据库分区总是视NULL值小于任何的一个非NULL值。

对于RANGE分区，如果向分区列插入NULL值，则MySQL数据库会将该值插入最左边的分区。

在LIST分区下要使用NULL值，则必须显式地指出哪个分区中放入NULL值，否则会报错。

HASH和KEY分区对于NULL的处理方式和RANGE分区、LIST分区不一样，任何分区函数都会将含有NULL值的记录返回为0。

## 分区和性能

数据库的应用分为两类：一类是OLTP（在线事务处理），如Blog、电子商务、网络游戏等；另一类是OLAP（在线分析处理），如数据仓库、数据集市。

对于OLAP的应用，分区的确是可以很好地提高查询的性能，因为OLAP应用大多数查询需要频繁地扫描一张很大的表。如果按照时间戳进行分区，则只需要扫描相应的分区即可。

**然而对于OLTP的应用，分区应该非常小心。在这种应用中，通常不可能获取一张大表中10%的数据，大部分都是通过索引返回几条记录即可。而根据B+树索引的原理，对于一张大表，一般的B+树需要2~3次的磁盘I/O。因此B+树可以很快地完成操作，不需要分区的帮助，并且设计不好的分区会带来严重的性能问题**。

在某行生产环境出现这样一种现象，select from t1 where uniq\_key=’’;按理说这种唯一索引查询效率应该非常高，但是实际却异常慢，最终分析发现这个表是30个分区的分区表，且where条件中没有分区键，导致扫描30个分区（串行），时间极慢。

## 表和分区间交换数据

要使用ALTER TABLE…EXCHANGE PARTITION语句，必须满足下面的条件：

1. 要交换的表需和分区表有着相同的表结构，但是表不能含有分区；
2. 在非分区表中的数据必须在交换的分区定义内；
3. 被交换的表中不能含有外键，或者其他的表含有对该表的外键引用；
4. 除了需要ALTER、INSERT和CREATE权限外，还需要DROP的权限外，

两个小的细节需要注意：

1. 使用该语句时，不会触发交换表和被交换表上的触发器；
2. AUTO\_INCREMENT列将被重置。

# 应用场景

对于上亿行或者更大数量的普通表清理，只能采用DELETE的方式，该方式不但性能差，而且无法真正回收空间。分区表可以通过删除分区等方式，对历史数据进行清理的同时数据文件也做了回收，真正释放了空间、而且效率很高。

很多互联网公司不建议用MySQL分区表，那么我个人的见解是：

对于特定场景是可以考虑采用分区表，如历史数据有明确的分区范围，访问不跨分区，极少的变化操作，查询语句逻辑简单，无性能瓶颈。

对于Oracle这些商业数据库，由于商业授权导致横向扩展成本较高，且分区表功能稳定，因此通过硬件扩展和分区来承担大数据量带来的负载。

对于MySQL开源数据库，企业有资源有能力将很多需求迁移到数据库外通过代码逻辑或者其它替代方式实现，因此更追求MySQL使用过程中的简单、稳定、可靠，且通过增加服务器以及分库分表更能有效处理数据量爆炸式增长带来的性能问题。

因此个人不建议大量使用MySQL分区表，尤其是在重要的业务上。