# 概述

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。

## 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；
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。

**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。

## 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。

# 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。

## KEY分区

根据MySQL数据库提供的散列函数来进行分区。

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);

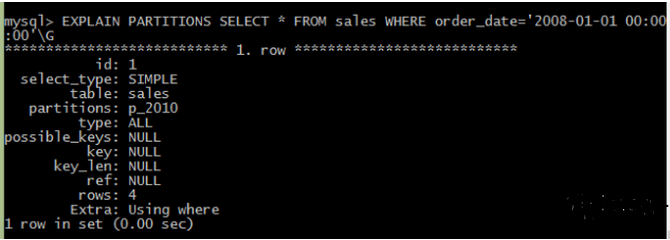
INSERT INTO sales (order\_date) VALUES (2011-01-01);

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

## 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**。

## 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。