## 7.1 分区表

MySQL分区意味着索引也是按照分区的子表定义的，而没有全局索引。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。

分区本身也有一些限制：

\* 一个表最多只能有1024个分区。

\* 在MySQL5.1中，分区表达式必须是整数，或者是返回整数的表达式。在MySQL5.5中，某些场景中可以直接使用列来进行分区。

\* 如果分区字段中有主键或者唯一索引的列，那么所有主键列和唯一索引列都必须包含进来。

\* 分区表中无法使用外键约束

### 7.1.1 分区表的原理

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上加上各自加上一个完全相同的索引。

分区表上的操作按照下面的操作逻辑进行：

select查询

当查询一个分区表的时候，分区层先打开并锁住所有底层表，优化器先判断是否可以过滤部分分区，然后再调用对应的存储引擎接口访问各个分区的数据。

insert操作

当写入一条记录时，分区层先打开并锁住所有底层表，然后确定哪个分区接收这条记录，再将记录写入对应底层表。

delete操作

当删除一条记录时，分区层先打开并锁住底层表，然后确定数据对应的分区，最后对相应底层表进行删除操作。

update操作

当更新一条记录时，分区层先打开并锁住所有底层表，MySQL先确定需要更新的记录在哪个分区，然后取出数据并更新，再判断更新后的数据应该放在哪个分区，最后对底层表进行写入操作，并对源数据所在的底层表进行删除操作。

### 7.1.2 分区表的类型

MySQL支持多种分区表。我们看到最多的是根据范围进行分区，每个分区存储落在某个范围的记录，分区表达式可以是列，也可以是包含列的表达式。例如，下表就可以将每一年的销售额存放在不同的分区里：



PARTITION分区子句中可以使用各种函数。但有一个要求，表达式返回的值要是一个确定的整数，而不能是常数。

MySQL还支持键值、哈希和列表分区，这其中有些还支持子分区，不过在生产环境中很少见到。在MySQL5.5中，还可以使用RANGE COLUMNS类型的分区。

### 7.1.3 如何使用分区表

在数据量超大的时候(比如10T)，B-Tree索引就无法起作用了。除非是索引覆盖查询。

为了保证大数据量的可扩展性，一般有下面两个策略：

**全量扫描数据，不要任何索引。**

可以使用简单的分区方式存放表，不要任何索引，根据分区的规则大致定位需要的数据位置。只要能够使用 where条件，将需要的数据限制在少量分区中，则效率是很高的。

**索引数据，并分离热点**

如果数据有明显的“热点”，并且除了这部分数据，其他数据很少被访问到，那么可以将这部分热点数据单独存放在一个分区中，

### 7.1.4 什么情况下会出问题

上面我们介绍的两个分区策略都基于两个非常重要的假设：查询都能够过滤掉很多额外的分区、分区本身并不会带来很多额外的代价。

**NULL值会使分区过滤无效**

分区表达式的值可以是NULL：第一个分区是一个特殊分区。假设按照PARTITION BY RANGE YEAR(order\_date)分区，那么所有order\_date为NULL或者是一个非法值的时候，记录都会被存放到第一个分区。现在假设有下面的查询：where order\_date between “2012-01-01” and “2012-01-31”。实际上，MySQL会检查两个分区：它会先检查2012这个分区，同时还会检查这个表的第一个分区。检查第一个分区是因为YEAR()函数在接收非法值的时候可能会返回，那么这个值就可能会返回到第一个分区。如果第一个分区非常大，特别是当使用“全量扫描数据，不要任何索引”的策略时，代价会非常大。

**分区列和索引列不匹配**

如果定义的索引列和分区列不匹配，会导致查询无法进行分区过滤。假设在列a上定义了索引，而在列b上进行分区。因为每个分区都有其独立的索引，所以扫描列b上的索引就需要扫描每一个分区内对应的索引。要避免这个问题，应该避免建立和分区列不匹配的索引。

**选择分区的成本可能很高**

**打开并锁住所有底层表的成本可能很高**

维护分区的成本可能很高