MySQL高级特性

# 分区表

## 分区概述

在MySQL中，InnoDB存储引擎长期支持表空间的概念，并且MySQL服务器甚至在分区引入之前，就能配置为存储不同的数据库使用不同的物理路径。

分区(partion)更进一步，它允许你通过设置各种规则将一个表的各个分区跨文件系统存储。实际上，不同位置的不同表分区是作为一个单独的表来存储的。用户所选择的、实现数据分割的规则被称为分区函数(partioning function)。

目前MySQL支持的分区如下：

* RANGE分区：基于属于一个给定连续区间的列值，把多行分配给分区。
* LIST分区：类似于按RANGE分区，区别在于LIST分区是基于列值匹配一个离散值集合中的某个值来进行选择。
* HASH分区：基于用户定义的表达式的返回值来进行选择的分区，该表达式使用将要插入到表中的这些行的列值进行计算。这个函数可以包含MySQL 中有效的、产生非负整数值的任何表达式。
* KEY分区：类似于按HASH分区，区别在于KEY分区只支持计算一列或多列，且MySQL 服务器提供其自身的哈希函数。必须有一列或多列包含整数值。

最常见是的水平分区(horizontal partitioning)，也就是将表的不同的元组分配到不同的物理分区上。目前，MySQL 5.1还不支持垂直分区(vertical pa

rtitioning)，即将表的不同列分配到不同的物理分区。在MySQL 5.1中，同一个表的各个分区必须使用相同的存储引擎。要为某个分区表配置一个专门的存储引擎，必须且只能使用[STORAGE] ENGINE 选项，这如同为非分区表配置存储引擎一样。但是，必须记住[STORAGE] ENGINE（和其他的表选项）必须列在用在CREATE TABLE语句中的其他任何分区选项之前。

注意：MySQL不支持全局分区，将索引也是在各自的分区中。另外，无论创建何种类型的分区，分区列必须是唯一索引的一个组成部分；如果分区表创建过程中没有主键和唯一索引，可以指定任意列作为分区列。最后，对于千万级别的数据不需要建立分区，建立分区不恰当反而会增加io次数（2,3\*分区数），建立一个好的索引来解决性能问题。

## 分区类型

### RANGE分区

|  |
| --- |
| #通过连续的id来进行分区，**MAXVALUE**指定最大值  **CREATE** **TABLE** `log\_flow` (  `id` **INT** **NOT** **NULL** **AUTO\_INCREMENT**,  `json` **TEXT** **NOT** **NULL**,  `ts` **TIMESTAMP** **NOT** **NULL** **DEFAULT** **CURRENT\_TIMESTAMP**,  `host` **VARCHAR**(200) **DEFAULT** **NULL** **COMMENT** '主机名',  `thread` **VARCHAR**(200) **DEFAULT** **NULL** **COMMENT** '线程名',  `body` **TEXT** **COMMENT** '主体',  `time` **VARCHAR**(50) **DEFAULT** **NULL** **COMMENT** '日志产生时间',  `response\_time` **BIGINT**(20) **DEFAULT** **NULL** **COMMENT** '响应时间',  **PRIMARY** **KEY** (`id`)  )  **ENGINE**=**InnoDB**  **PARTITION** **BY** **RANGE** (id)(  **PARTITION** p0 **VALUES** **LESS** **THAN** (100000),  **PARTITION** p1 **VALUES** **LESS** **THAN** (200000),  **PARTITION** p2 **VALUES** **LESS** **THAN** **MAXVALUE**); |
| #根据年份进行分区。  **CREATE** **TABLE** `log\_flow\_1` (  `id` **INT** **NOT** **NULL**,  `json` **TEXT** **NOT** **NULL**,  `ts` **DATETIME** **NOT** **NULL** **DEFAULT** **CURRENT\_TIMESTAMP**,  `host` **VARCHAR**(200) **DEFAULT** **NULL** **COMMENT** '主机名',  `thread` **VARCHAR**(200) **DEFAULT** **NULL** **COMMENT** '线程名',  `body` **TEXT** **COMMENT** '主体',  `time` **VARCHAR**(50) **DEFAULT** **NULL** **COMMENT** '日志产生时间',  `response\_time` **BIGINT**(20) **DEFAULT** **NULL** **COMMENT** '响应时间'  ) **ENGINE**=**InnoDB**  **PARTITION** **BY** **RANGE**(**YEAR**(ts))(  **PARTITION** p2009 **VALUES** **LESS** **THAN** (2010),  **PARTITION** p2010 **VALUES** **LESS** **THAN** (2011),  **PARTITION** p2011 **VALUES** **LESS** **THAN** (2012));  #删除分区  drop table log\_flow\_1 drop partition p2010; |
| #根据年月进行分区  **CREATE** **TABLE** `log\_flow\_2` (  `id` **INT** **NOT** **NULL**,  `json` **TEXT** **NOT** **NULL**,  `ts` **DATETIME** **NOT** **NULL** **DEFAULT** **CURRENT\_TIMESTAMP**,  `host` **VARCHAR**(200) **DEFAULT** **NULL** **COMMENT** '主机名',  `thread` **VARCHAR**(200) **DEFAULT** **NULL** **COMMENT** '线程名',  `body` **TEXT** **COMMENT** '主体',  `time` **VARCHAR**(50) **DEFAULT** **NULL** **COMMENT** '日志产生时间',  `response\_time` **BIGINT**(20) **DEFAULT** **NULL** **COMMENT** '响应时间'  ) **ENGINE**=**InnoDB**  **PARTITION** **BY** **RANGE**(**to\_days**(ts))(  **PARTITION** p201001 **VALUES** **LESS** **THAN** (**to\_days**('2010-02-01')),  **PARTITION** p201002 **VALUES** **LESS** **THAN** (**to\_days**('2010-03-01')),  **PARTITION** p201003 **VALUES** **LESS** **THAN** (**to\_days**('2010-04-01')));  #添加分区  ALTER TABLE log\_flow\_2 ADD PARTITION(PARTITION p201004 VALUES LESS THAN (TO\_DAYS('2010-05-01'))); |

注意：对于RANGE分区，如果向分区插入NULL值，MySQL会将该值放入最左边的分区。

### LIST分区

|  |
| --- |
| # LIST分区和RANGE分区十分相似，只是LIST是离散的值  **CREATE** **TABLE** `test\_list` (  `id` **INT** **NOT** **NULL**  ) **ENGINE**=**InnoDB**  **PARTITION** **BY** **LIST**(id)(  **PARTITION** p0 **VALUES** **IN** (**NULL**,1,3,5),  **PARTITION** p1 **VALUES** **IN** (2,4,6)); |

注意：对于LIST分区下使用NULL值，则应该显示的指出哪个区放入NULL值，否则会报错。

### HASH分区

HASH分区的目的是将数据均匀的分布在各个分区中，保证分区的数量大致都是一样。RANGE和LIST分区需要制定分区名，而在HASH中，MySQL自动完成。

|  |
| --- |
| **CREATE** **TABLE** `test\_hash` (  `id` **INT** **NOT** **NULL**,  `create\_data` **DATETIME**  ) **ENGINE**=**InnoDB**  **PARTITION** **BY** **HASH**(**YEAR**(create\_data))  **PARTITIONS** 4;  注意：对于HASH分区下使用NULL值，会将其返回0. |

### KEY分区

KEY分区和HASH分区相似，不同之处在于HASH分区使用用户定义的函数进行分区，KEY分区使用MYSQL提供的函数进行分区。

|  |
| --- |
| **CREATE** **TABLE** `test\_key` (  `id` **INT** **NOT** **NULL**,  `create\_data` **DATETIME**  ) **ENGINE**=**InnoDB**  **PARTITION** **BY** **KEY**(create\_data)  **PARTITIONS** 4;  注意：对于HASH分区下使用NULL值，会将其返回0. |

### COLUMNS分区

前面介绍的四种分区：RANGE,LIST,HASH,KEY必须是整数类型。如果不是整数类型，要通过函数转化为整数。COLUMNS(MYSQL5.5)可以直接使用非整数类型进行分区，不需要转化成整数类型。COLUMNS支持的类型有：

* 所有的整数类型
* 日期类型：DATETIME,DATE
* 字符类型：CHAR,VARCHAR,BINARY,VARBINARY

|  |
| --- |
| #需要columns要修饰range或者list  **CREATE** **TABLE** `test\_columns` (  `id` **INT** **NOT** **NULL**,  `create\_data` **DATETIME**  ) **ENGINE**=**InnoDB**  **PARTITION** **BY** **RANGE** **COLUMNS**(create\_data) (  **PARTITION** p201001 **VALUES** **LESS** **THAN** ('2010-02-01'),  **PARTITION** p201002 **VALUES** **LESS** **THAN** ('2010-03-01')); |

## 在表和分区之间交换数据

|  |
| --- |
| 使用ALTER TABLE … EXCHAGE PARTITION 语句条件：   * 交换的表和分区表有相同的表结构，但是表不能含有分区。 * 被交换的表不能有外键关系   语句：ALTER TABLE 分区表\_name EXCHAGE PARTITION 分区\_name WITH TABLE tb\_name  注意：该语句不会触发触发器，并且会重置auto\_increment列 |

# 查询缓存

## 查询缓存的工作流程

MySQL查询缓存可以跳过SQL解析优化查询等阶段,直接返回缓存结果给用户,查询缓存的工作流程如下:

* 服务器接收SQL,以SQL和一些其他条件为key查找缓存表(额外性能消耗)
* 如果找到了缓存,则直接返回缓存(性能提升)
* 如果没有找到缓存,则执行SQL查询,包括原来的SQL解析,优化等.
* 执行完SQL查询结果以后,将SQL查询结果存入缓存表(额外性能消耗)

命中条件：：缓存存在一个hash表中,通过查询SQL,查询数据库,客户端协议等作为key.在判断是否命中前,MySQL不会解析SQL,而是直接使用SQL去查询缓存,SQL任何字符上的不同,如空格,注释,都会导致缓存不命中。如果查询中有不确定数据,例如CURRENT\_DATE()和NOW()函数,那么查询完毕后则不会被缓存.所以,包含不确定数据的查询是肯定不会找到可用缓存的

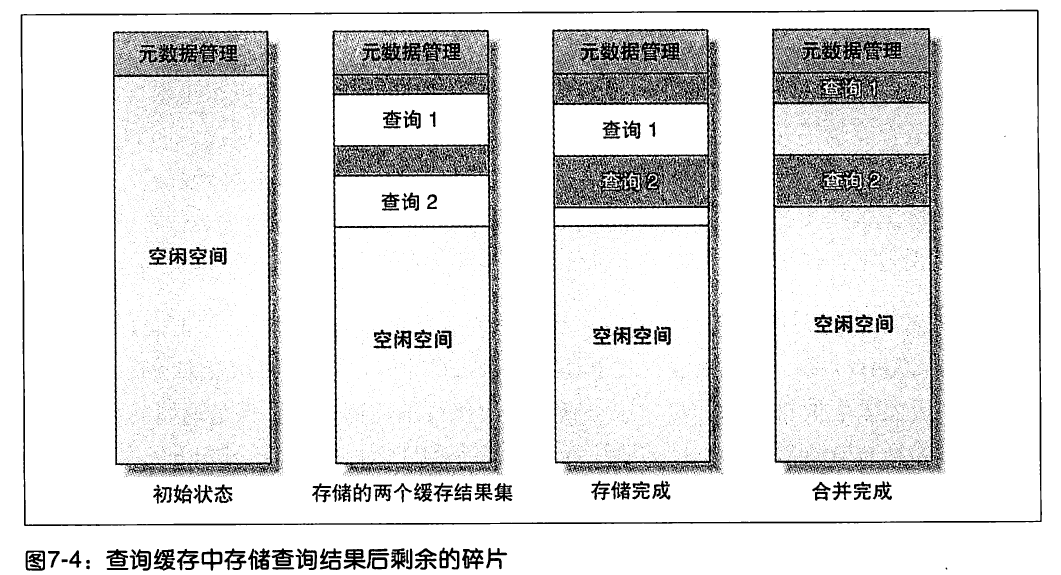
缓存失效：查询中包含任何用户自定义函数，存储过程，用户变量，临时表，mysql库中的系统表或者任何包含列级别权限的表，都不会缓存。另外，当某个表正在写入数据,则这个表的缓存(命中检查,缓存写入等)将会处于失效状态.在Innodb中,如果某个事务修改了表,则这个表的缓存在事务提交前都会处于失效状态,在这个事务提交前,这个表的相关查询都无法被缓存.

## 缓存的内存管理

缓存会在内存中开辟一块内存(query\_cache\_size)来维护缓存数据,其中有大概40K的空间是用来维护缓存的元数据的,例如空间内存,数据表和查询结果的映射,SQL和查询结果的映射等.

MySQL将这个大内存块分为小的内存块(query\_cache\_min\_res\_unit),每个小块中存储自身的类型,大小和查询结果数据,还有指向前后内存块的指针.

MySQL需要设置单个小存储块的大小,在SQL查询开始(还未得到结果)时就去申请一块空间,所以即使你的缓存数据没有达到这个大小,也需要用这个大小的数据块去存(这点跟Linux文件系统的Block一样).如果结果超出这个内存块的大小,则需要再去申请一个内存块.当查询完成发现申请的内存块有富余,则会将富余的空间释放掉,这就会造成内存碎片问题,见下图



此处查询1和查询2之间的空白部分就是内存碎片,这部分空闲内存是有查询1查询完以后释放的,假设这个空间大小小于MySQL设定的内存块大小,则无法再被使用,造成碎片问题。

在查询开始时申请分配内存Block需要锁住整个空闲内存区,所以分配内存块是非常消耗资源的.注意这里所说的分配内存是在MySQL初始化时就开辟的那块内存上分配的。

## 缓存的使用时机

衡量打开缓存是否对系统有性能提升是一个很难的话题

* 通过缓存命中率判断, 缓存命中率 = 缓存命中次数 (Qcache\_hits) / 查询次数 (Com\_select)
* 通过缓存写入率, 写入率 = 缓存写入次数 (Qcache\_inserts) / 查询次数 (Qcache\_inserts)
* 通过 命中-写入率 判断, 比率 = 命中次数 (Qcache\_hits) / 写入次数 (Qcache\_inserts), 高性能MySQL中称之为比较能反映性能提升的指数,一般来说达到3:1则算是查询缓存有效,而最好能够达到10:1

## 缓存配置参数

query\_cache\_type: 是否打开缓存。OFF: 关闭；ON: 总是打开；DEMAND: 只有明确写了SQL\_CACHE的查询才会吸入缓存

query\_cache\_size: 缓存使用的总内存空间大小，单位是字节,这个值必须是1024的整数倍,否则MySQL实际分配可能跟这个数值不同(感觉这个应该跟文件系统的blcok大小有关)

query\_cache\_min\_res\_unit: 分配内存块时的最小单位大小

query\_cache\_limit: MySQL能够缓存的最大结果,如果超出,则增加 Qcache\_not\_cached的值,并删除查询结果

query\_cache\_wlock\_invalidate: 如果某个数据表被锁住,是否仍然从缓存中返回数据,默认是OFF,表示仍然可以返回

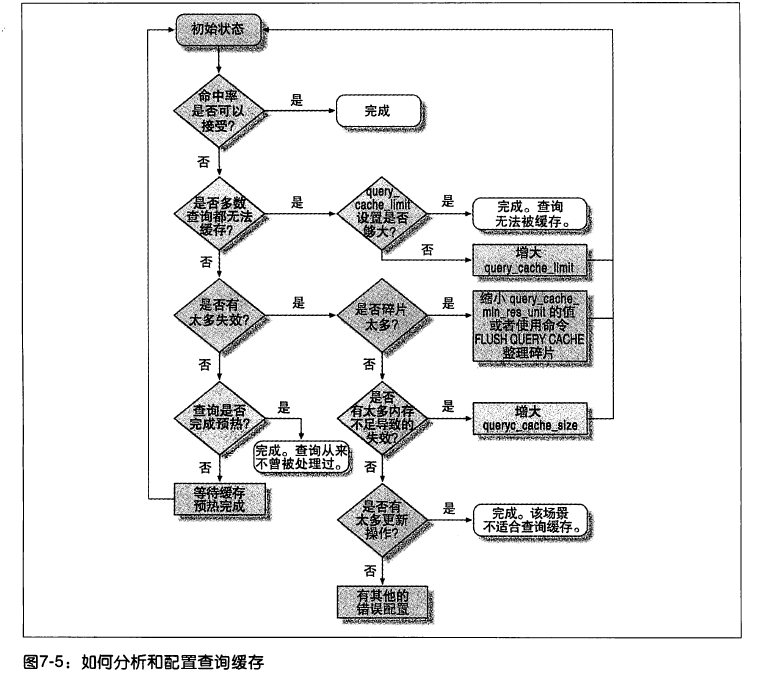
GLOBAL STAUS 中 关于 缓存的参数解释:

* Qcache\_free\_blocks: 缓存池中空闲块的个数
* Qcache\_free\_memory: 缓存中空闲内存量
* Qcache\_hits: 缓存命中次数
* Qcache\_inserts: 缓存写入次数
* Qcache\_lowmen\_prunes: 因内存不足删除缓存次数
* Qcache\_not\_cached: 查询未被缓存次数,例如查询结果超出缓存块大小,查询中包含可变函数等
* Qcache\_queries\_in\_cache: 当前缓存中缓存的SQL数量
* Qcache\_total\_blocks: 缓存总block数

减少碎片策略**：** 选择合适的block大小**；**使用 FLUSH QUERY CACHE 命令整理碎片；这个命令在整理缓存期间,会导致其他连接无法使用查询缓存

PS: 清空缓存的命令式 RESET QUERY CACHE

查询缓存问题分析：



## InnoDB与查询缓存

Innodb会对每个表设置一个事务计数器,里面存储当前最大的事务ID.当一个事务提交时,InnoDB会使用MVCC中系统事务ID最大的事务ID跟新当前表的计数器。

只有比这个最大ID大的事务能使用查询缓存,其他比这个ID小的事务则不能使用查询缓存。另外,在InnoDB中,所有有加锁操作的事务都不使用任何查询缓存。