### 优化小手段：

**1.EXPLAIN命令**

如果在SELECT语句前放上关键词EXPLAIN，MySQL将解释它如何处理SELECT，提供有关表如何联接和联接的次序。

借助于EXPLAIN，可以知道什么时候必须为表加入索引以得到一个使用索引来寻找记录的更快的SELECT。

**2.php代替复杂处理**

Sql要做一些很复杂的处理的话，可以分出来交给后端语言去完成。特别是一些case when 什么的。

**3.字段设计为非空**

非null字段的处理要比null字段的处理高效些！且不需要判断是否为null。MySQL中每条记录都需要额外的存储空间，表示每个字段是否为null。因此通常使用特殊的数据进行占位，比如int not null default 0、string not null default ‘’。就是每个字段都给他默认值，给个0 好，‘’也好。因为判断数据是否Null比前者慢。

**4.** **order by字段建立索引**

当我们使用order by将查询结果按照某个字段排序时，如果该字段没有建立索引，那么执行计划会将查询出的所有数据使用外部排序（将数据从硬盘分批读取到内存使用内部排序，最后合并排序结果），这个操作是很影响性能的，因为需要将查询涉及到的所有数据从磁盘中读到内存（如果单条数据过大或者数据量过多都会降低效率），更无论读到内存之后的排序了。

但是如果我们对该字段建立索引alter table 表名 add index(字段名)，那么由于索引本身是有序的，因此直接按照索引的顺序和映射关系逐条取出数据即可。而且如果分页的，那么只用取出索引表某个范围内的索引对应的数据，而不用像上述那取出所有数据进行排序再返回某个范围内的数据。（从磁盘取数据是最影响性能的）

**5.** **join表的On字段建立索引**

对join语句匹配关系（on）涉及的字段建立索引能够提高效率

**6.** **索引覆盖，对常查询的字段建立索引**

什么是索引覆盖，就是一条sql里面用到的字段，比如select后面的，order by 的，group by的等等用到的字段，只要有一个没建索引的，就会进行全表扫描。

所以我们进可能对所有常用的数据建立索引，但是不要想着为每个字段建立索引，那跟没建也没啥区别，还多了索引记录的更慢了。

注：状态值（例如性别、支付状态等状态值字段）不容易触发到索引，就可以不建了

**7.** **like查询，不能以通配符开头**

例如，在title字段建立了索引的前提下。

select \* from article where title like 'mysql%';会触发索引。

select \* from article where title like '%mysql%';不会触发索引。

**8.** **在使用InnoDB存储引擎时，如果没有特别的需要，请永远使用一个与业务无关的自增字段作为主键**

InnoDB使用聚集索引，各条数据记录按主键顺序存放。

因此每当有一条新的记录插入时，MySQL会根据其主键将其插入适当的节点和位置。

如果使用非自增主键（如果身份证号或学号等），插入数据时，MySQL不得不为了将新记录插到合适位置而移动数据，甚至目标页面可能已经被回写到磁盘上而从缓存中清掉，此时又要从磁盘上读回来，这增加了很多开销，同时频繁的移动、分页操作造成了大量的碎片。

**9.** **最左前缀匹配原则**

mysql会一直从左向右匹配直到遇到范围查询(>、<、between、like)就停止匹配。

比如a = 1 and b = 2 and c > 3 and d = 4 如果建立(a,b,c,d)顺序的索引，d是用不到索引的，如果建立(a,b,d,c)的索引则都可以用到，a,b,d的顺序可以任意调整。

所以，写语句时，能命中索引的绝对条件写在前面，范围查询写在后面。

**10.** **索引列不能参与计算**

例如，在title字段建立了索引的前提下。

from\_unixtime(create\_time) = ’2014-05-29’;是错误的。

create\_time = unix\_timestamp(’2014-05-29’);才是正确的。

<https://blog.csdn.net/weixin_38405253/article/details/100193675> <https://blog.csdn.net/y277an/article/details/100515951>

### 查询缓存：

**描述：**

查询缓存就是缓存select语句的查询结果。

**开启方法：**

在配置文件中开启缓存

windows上是my.ini，linux上是my.cnf

在[mysqld]段中配置query\_cache\_type：

0：不开启

1：开启，默认缓存所有，需要在SQL语句中增加select sql-no-cache提示来放弃缓存

2：开启，默认都不缓存，需要在SQL语句中增加select sql-cache来主动缓存（常用）

更改配置后需要重启以使配置生效，重启后可通过show variables like ‘query\_cache\_type’;来查看：

**缓存区大小配置：**

query\_cache\_size

**使用方法：**

将查询结果缓存

select sql\_cache \* from user;

重置缓存

reset query cache;

**注意事项：**

1. 当某个数据表改动时，牵涉到该表的所有缓存会失效。（所以其实这个mysql缓存也不是那么好用，因为实际中的查询操作都需要联表，而且常查的表通常也常增删改的操作）。如果开启了总是缓存的话，这经常又存又删的，耗费的系统资源很大，不开好过开。所以一般就算开也是手动开。
2. 缓存是以SQL语句为key存储的，因此即使SQL语句功能相同，但如果多了一个空格或者大小写有差异都会导致匹配不到缓存。

### 分区：

注：

数据表分区和分表是不同的！！！

服务端的表分区对于客户端是透明的，客户端还是照常插入数据，但服务端会按照分区算法分散存储数据。

分区依据的字段必须是主键的一部分，分区是为了快速定位数据，因此该字段的搜索频次较高应作为强检索字段，否则依照该字段分区毫无意义

**意义：**

当数据量较大时（一般千万条记录级别以上），MySQL的性能就会开始下降，这时我们就需要将数据分散到多组存储文件，**保证其单个文件的执行效率**。

**方案：**

方案主要分为3种，按余数分区、按范围分区、按值分区

**1、按余数分区**

最常见的分区方案是按id分区，如下将id的哈希值对10取模将数据均匀分散到10个.ibd存储文件中：

create table article(

id int auto\_increment PRIMARY KEY,

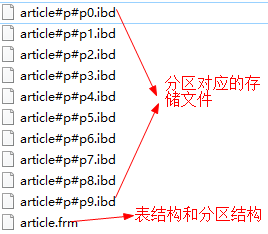
title varchar(64),

content text

)PARTITION by HASH(id) PARTITIONS 10

**注**： 分区值为Int时用hash，字符串时用key（PARTITION by KEY(id) PARTITIONS 10），性质上是一样的，只不过key是处理字符串的，比hash()多了一步从字符串中计算出一个整型在做取模操作。

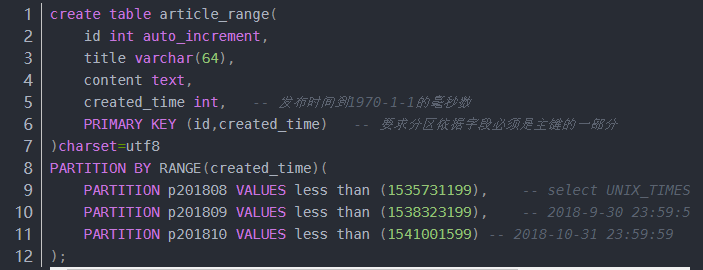
查看data目录：

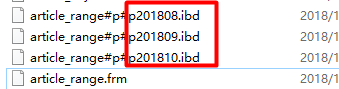


**2、按范围分区**

是一种条件分区算法，按照数据大小范围分区（将数据使用某种条件，分散到不同的分区中）。

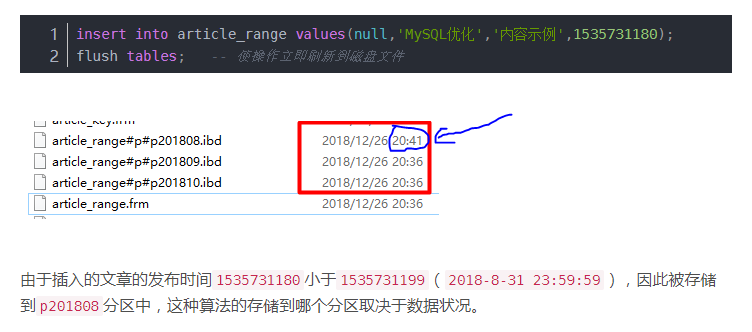
如下，按文章的发布时间将数据按照2018年8月、9月、10月分区存放：





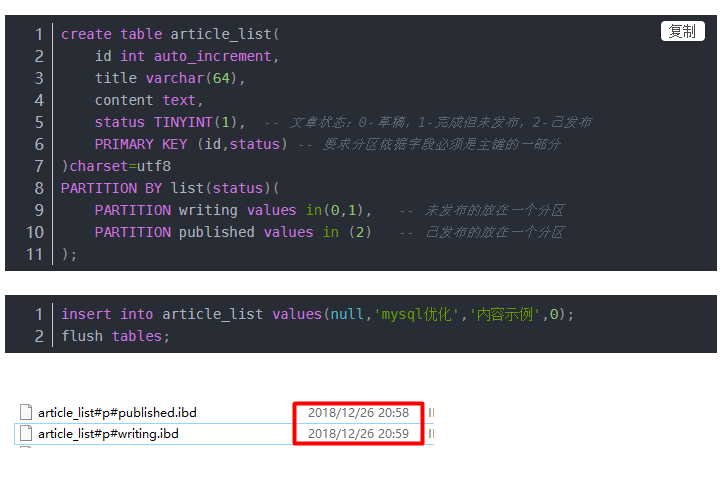
**注意：**条件运算符只能使用less than，这以为着较小的范围要放在前面，比如上述p201808,p201819,p201810分区的定义顺序依照created\_time数值范围从小到大，不能颠倒。

**Demo：**



**3、按值分区**

也是一种条件分区，按照列表值分区（in (值列表)）。



## 分区的使用

当数据表中的数据量很大时，分区带来的效率提升才会显现出来。

只有检索字段为分区字段时，分区带来的效率提升才会比较明显。因此，分区字段的选择很重要，并且业务逻辑要尽可能地根据分区字段做相应调整（尽量使用分区字段作为查询条件）。

# 水平分割和垂直分割

水平分割：通过建立结构相同的几张表分别存储数据

垂直分割：将经常一起使用的字段放在一个单独的表中，分割后的表记录之间是一一对应关系。

## 分表原因

* 为数据库减压
* 分区算法局限
* 数据库支持不完善（5.1之后mysql才支持分区操作）

## id重复的解决方案

* 借用第三方应用如memcache、redis的id自增器
* 单独建一张只包含id一个字段的表，每次自增该字段作为数据记录的id