# Mysql索引

# 掌握：B+tree的数据结构特征、叶子节点和非叶子节点存储的数据内容、索引优化。

### 索引

索引是对数据库表 中一个或多个列的值进行排序的结构。与在表中搜索所有的行相比，索引用指针 指向存储在表中指定列的数据值，然后根据指定的次序排列这些指针，有助于更快地获取信息。通常情 况下 ，只有当经常查询索引列中的数据时 ，才需要在表上创建索引。索引将占用磁盘空间，并且影响数 据更新的速度。但是在多数情况下 ，索引所带来的数据检索速度优势大大超过它的不足之处。

在 MySQL 中，主要有四种类型的索引，分别为： B-Tree 索引（技术上说B+Tree）， Hash 索引， Fulltext 索引和 R-Tree 索引。

### 索引类型

* 按数据结构划分

Mysql目前主要有以下几种索引类型：FULLTEXT，HASH，BTREE，RTREE。

**1. FULLTEXT**

即为全文索引，目前只有MyISAM引擎支持。其可以在CREATE TABLE ，ALTER TABLE ，CREATE INDEX 使用，不过目前只有 CHAR、VARCHAR ，TEXT 列上可以创建全文索引。

全文索引并不是和MyISAM一起诞生的，它的出现是为了解决WHERE name LIKE “%word%"这类针对文本的模糊查询效率较低的问题。

**2. HASH**

由于HASH的唯一（几乎100%的唯一）及类似键值对的形式，很适合作为索引。

HASH索引可以一次定位，不需要像树形索引那样逐层查找,因此具有极高的效率。但是，这种高效是有条件的，即只在“=”和“in”条件下高效，对于范围查询、排序及组合索引仍然效率不高。

**3. BTREE**

BTREE索引就是一种将索引值按一定的算法，存入一个树形的数据结构中（二叉树），每次查询都是从树的入口root开始，依次遍历node，获取leaf。这是MySQL里默认和最常用的索引类型。

**4. RTREE**

RTREE在MySQL很少使用，仅支持geometry数据类型，支持该类型的存储引擎只有MyISAM、BDb、InnoDb、NDb、Archive几种。

相对于BTREE，RTREE的优势在于范围查找。

* 按逻辑划分

primary：唯一索引，不允许为null。

key：普通非唯一索引。

unique：表示唯一的，不允许重复的索引，可以为null。

fulltext: 表示全文搜索的索引。 FULLTEXT用于搜索很长一篇文章的时候，效果最好。用在比较短的文本，如果就一两行字的，普通的INDEX 也可以。

spatial：空间索引。

**注意：不同的索引，它的存储结构是不一样的，这里我们主要分析一下B-tree索引的数据结构。**

### B+Tree

**m阶B+Tree特征如下**：

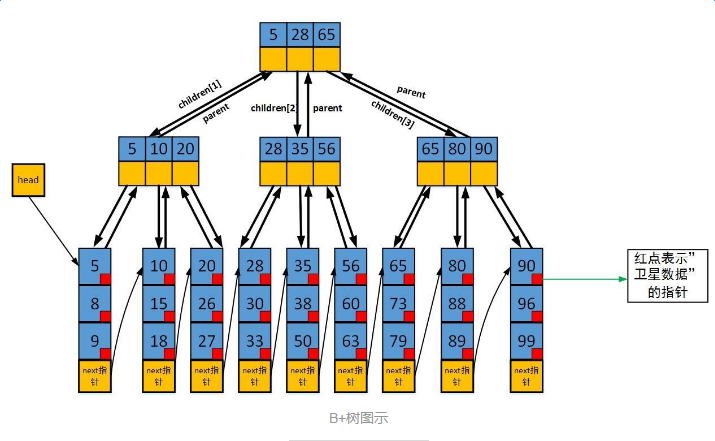
1、有m个子树的节点包含有m个元素（B-Tree中是m-1）

2、根节点和分支节点中不保存数据，只用于索引，所有数据都保存在叶子节点中。

3、所有分支节点和根节点都同时存在于子节点中，在子节点元素中是最大或者最小的元素。

4、叶子节点会包含所有的关键字，以及指向数据记录的指针，并且叶子节点本身是根据关键字的大小从小到大顺序链接。

### 图例



说明：

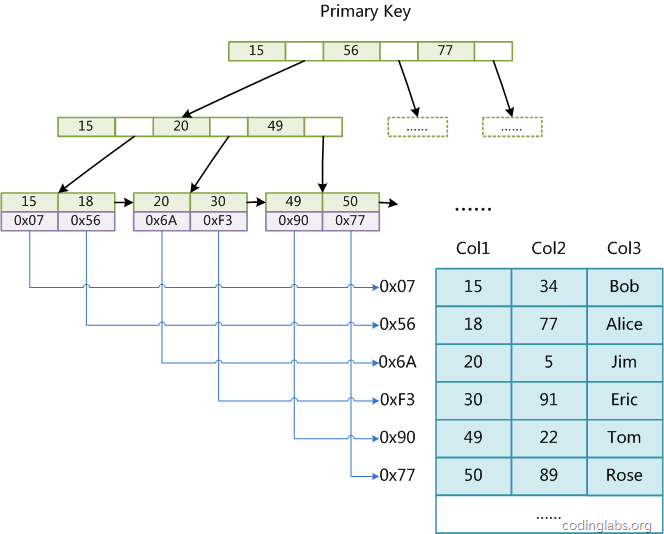
1、红点表示是指向卫星数据的指针，指针指向的是存放实际数据的磁盘页，卫星数据就是数据库中一条数据记录。

2、叶子节点中还有一个指向下一个叶子节点的next指针，所以叶子节点形成了一个有序的链表，方便遍历B+树。

### MyISAM索引实现：

1. 主键索引：

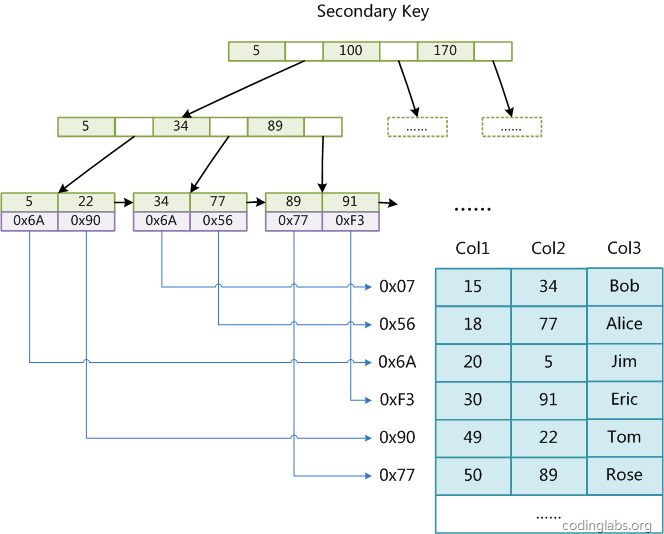
MyISAM引擎使用B+Tree作为索引结构，叶节点的data域存放的是数据记录的地址。下图是MyISAM主键索引的原理图：



这里设表一共有三列，假设我们以Col1为主键，图myisam1是一个MyISAM表的主索引（Primary key）示意。可以看出MyISAM的索引文件仅仅保存数据记录的地址。

辅助索引（Secondary key）

在MyISAM中，主索引和辅助索引（Secondary key）在结构上没有任何区别，只是主索引要求key是唯一的，而辅助索引的key可以重复。如果我们在Col2上建立一个辅助索引，则此索引的结构如下图所示：



同样也是一颗B+Tree，data域保存数据记录的地址。因此，MyISAM中索引检索的算法为首先按照B+Tree搜索算法搜索索引，如果指定的Key存在，则取出其data域的值，然后以data域的值为地址，读取相应数据记录。

### InnoDB索引实现

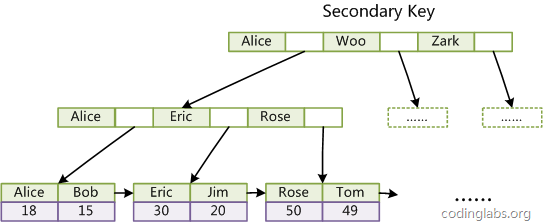
虽然InnoDB也使用B+Tree作为索引结构，但具体实现方式却与MyISAM截然不同.



(图inndb主键索引）是InnoDB主索引（同时也是数据文件）的示意图，可以看到叶节点包含了完整的数据记录。这种索引叫做聚集索引。因为InnoDB的数据文件本身要按主键聚集，所以InnoDB要求表必须有主键（MyISAM可以没有），如果没有显式指定，则MySQL系统会自动选择一个可以唯一标识数据记录的列作为主键，如果不存在这种列，则MySQL自动为InnoDB表生成一个隐含字段作为主键，这个字段长度为6个字节，类型为长整形。

InnoDB的辅助索引

InnoDB的所有辅助索引都引用主键作为data域。例如，下图为定义在Col3上的一个辅助索引：



    InnoDB 表是基于聚簇索引建立的。因此InnoDB 的索引能提供一种非常快速的主键查找性能。不过，它的辅助索引（Secondary Index， 也就是非主键索引）也会包含主键列，所以，如果主键定义的比较大，其他索引也将很大。如果想在表上定义 、很多索引，则争取尽量把主键定义得小一些。InnoDB 不会压缩索引。

不同存储引擎的索引实现方式对于正确使用和优化索引都非常有帮助，例如知道了InnoDB的索引实现后，就很容易明白为什么不建议使用过长的字段作为主键，因为所有辅助索引都引用主索引，过长的主索引会令辅助索引变得过大。再例如，用非单调的字段作为主键在InnoDB中不是个好主意，因为InnoDB数据文件本身是一颗B+Tree，非单调的主键会造成在插入新记录时数据文件为了维持B+Tree的特性而频繁的分裂调整，十分低效，而使用自增字段作为主键则是一个很好的选择。

### InnoDB索引和MyISAM索引的区别：

一是主索引的区别，InnoDB的数据文件本身就是索引文件。而MyISAM的索引和数据是分开的。

二是辅助索引的区别：InnoDB的辅助索引data域存储相应记录主键的值而不是地址。而MyISAM的辅助索引和主索引没有多大区别。

### 索引优化

### 优化步骤

1、通过show status 命令了解各种sql的执行效率

2、定位执行效率较低的SQL语句（通过慢查询记录+show processlist命令查看当前线程）

3、通过explain分析低效率的SQL语句的执行情况

4、确定问题并采取相应的优化措施。

具体实践步骤参考：<https://www.cnblogs.com/jian0110/p/9356347.html>

慢查询日志参考博文：

<https://blog.csdn.net/m_nanle_xiaobudiu/article/details/79288257>

explain使用说明

<https://www.cnblogs.com/yycc/p/7338894.html>

### 索引的好处

1、提高数据检索的效率，降低检索过程中必须要读取得数据量，降低数据库IO成本。

2、降低数据库的排序成本。因为索引就是对字段数据进行排序后存储的，如果待排序的字段与索引键字段一致，就在取出数据后不用再次排序了，因为通过索引取得的数据已满足排序要求。另外，分组操作是先排序后分组，所以索引同样可以省略分组的排序操作，降低内存与CPU资源的消耗。

### 索引的弊端

1、索引会增加写操作的成本

2、太多的索引会增加查询优化器的选择时间

3、索引要占用空间，随着数据量的不断增大，索引还会带来存储空间的消耗

### 优化建议

1. **前导模糊查询不能使用索引。**

例如下面 SQL 语句不能使用索引。

select \* fromdoc where title like '%XX'

而非前导模糊查询则可以使用索引，如下面的 SQL 语句。

select \* fromdoc where title like 'XX%'

1. **union、in、or 都能够命中索引，建议使用 in。**

union：能够命中索引。

示例代码如下：

select \* fromdoc where status=1

union all

select \* fromdoc where status=2

直接告诉 MySQL 怎么做，MySQL 耗费的 CPU 最少，但是一般不这么写 SQL。

in：能够命中索引。

示例代码如下：

select \* fromdoc where status in (1, 2)

查询优化耗费的 CPU 比 union all 多，但可以忽略不计，一般情况下建议使用 in

or：新版的 MySQL 能够命中索引。

示例代码如下：

select \* from doc where status = 1 or status = 2

查询优化耗费的 CPU 比 in 多，不建议频繁用 or。

1. **负向条件查询不能使用索引，可以优化为 in 查询。**

负向条件有：!=、<>、not in、not exists、not like 等。

例如下面代码：

select \* fromdoc where status != 1 and status != 2

可以优化为 in 查询：

select \* fromdoc where status in (0,3,4)

1. **联合索引最左前缀原则（又叫最左侧查询）**

如果在(a,b,c)三个字段上建立联合索引，那么它能够加快 a | (a,b) | (a,b,c) 三组查询速度。 Select \* from table where b=””(不走索引)

1. **强制类型转换会全表扫描**

如果 phone 字段是 varchar 类型，则下面的 SQL 不能命中索引。

select \* fromuser where phone=13800001234

可以优化为：select \* fromuser where phone='13800001234'

1. **范围列可以用到索引（联合索引必须是最左前缀）。**

范围条件有：<、<=、>、>=、between等。

范围列可以用到索引（联合索引必须是最左前缀），但是范围列后面的列无法用到索引，索引最多用于一个范围列，如果查询条件中有两个范围列则无法全用到索引。

mysql sql 中< 小于 > 大于 <= >= 这个根据实际查询数据来判断，如果全盘扫描速度比索引速度要快则不走索引 。

假如有联合索引 (empno、title、fromdate)，那么下面的 SQL 中 emp\_no 可以用到索引，而 title 和 from\_date 则使用不到索引。

select \* from employees.titles where emp\_no < 10010' and title='Senior Engineer'and from\_date between '1986-01-01' and '1986-12-31'

1. **应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描**
2. **应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：**

select id from t where num/2=100

1. **应尽量避免在where子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描**

select id from t where substring(name,1,3)='abc'

1. **GROUP BY 关键字优化**

select sex from employee where dep=123 group by sex having age >10;

1、group by 实质是先排序后分组，遵照索引的最佳左前缀。

2、where 高于having，能写在where限定的条件就不要去having去限定了。

注意：并不是所有索引对查询都有效，SQL是根据表中数据来进行查询优化的，当索引列有大量数据重复时，SQL查询可能不会去利用索引，如一表中有字段sex，male、female几乎各一半，那么即使在sex上建了索引也对查询效率起不了作用。

### Union和union all区别

Union，对两个结果集进行并集操作，不包括重复行，同时进行默认规则的排序；

Union All，对两个结果集进行并集操作，包括重复行，不进行排序；

从效率上说，UNION ALL 要比UNION快很多，所以，如果可以确认合并的两个结果集中不包含重复的数据的话，那么就使用UNION ALL。

### 强制和禁止索引

mysql强制使用索引:force index(索引名或者主键PRI)

select \* from table force index(ziduan1\_index) limit 2;(强制使用索引”ziduan1\_index”)

mysql禁止某个索引：ignore index(索引名或者主键PRI)

select \* from table ignore index(ziduan1\_index) limit 2;(禁止使用索引”ziduan1\_index”)

### 题目：

**1. 请问如下三条 SQL 该如何建立索引？**

where a=1 and b=1

where b=1

where b=1order by time desc

**MySQL 的查询优化器会自动调整 where 子句的条件顺序以使用适合的索引吗？**

回答：

第一问：建议建立两个索引，即 idxab(a,b) 和 idxbtime(b,time)。

第二问：MySQL 的查询优化器会自动调整 where 子句的条件顺序以使用适合的索引，对于上面的第一条 SQL，如果建立索引为 idxba(b,a) 也是可以用到索引的，不过建议 where 后的字段顺序和联合索引保持一致，养成好习惯。

### Mysql查看是否使用到索引

mysql数据库创建索引优化之后，在查询时想看下是否使用到索引，

使用explain查看执行计划：

|  |
| --- |
| mysql> explain SELECT \* FROM tb\_user  WHERE STATUS=1 limit 0,20;  +----+-------------+----------------+------------+------+----------------------+----------------------+---------+-------+-------+----------+-------+  | id | select\_type | table | partitions | type | possible\_keys | key | key\_len | ref | rows | filtered | Extra |  +----+-------------+----------------+------------+------+----------------------+----------------------+---------+-------+-------+----------+-------+  | 1 | SIMPLE | tb\_news\_online | NULL | ref | idx\_tb\_news\_online\_9 | idx\_tb\_news\_online\_9 | 5 | const | 99494 | 100 | NULL |  +----+-------------+----------------+------------+------+----------------------+----------------------+---------+-------+-------+----------+-------+  --------------------- |

table：显示这一行的数据是关于哪张表的

type：这是重要的列，显示连接使用了何种类型。从最好到最差的连接类型为const、eq\_reg、ref、range、 indexhe和ALL

possible\_keys：显示可能应用在这张表中的索引。如果为空，没有可能的索引。可以为相关的域从WHERE语句中选择一个合适的语句

key： 实际使用的索引。如果为NULL，则没有使用索引。很少的情况下，MYSQL会选择优化不足的索引。这种情况下，可以在SELECT语句中使用USE INDEX（indexname）来强制使用一个索引或者用IGNORE INDEX（indexname）来强制MYSQL忽略索引

key\_len：使用的索引的长度。在不损失精确性的情况下，长度越短越好

ref：显示索引的哪一列被使用了，如果可能的话，是一个常数

rows：MYSQL认为必须检查的用来返回请求数据的行数

Extra：关于MYSQL如何解析查询的额外信息。