链滴

mysql 的索引

作者：xiaokedamowang
原文链接：https://ld246.com/article/1628705854195
来源网站：链滴

mysql 索引

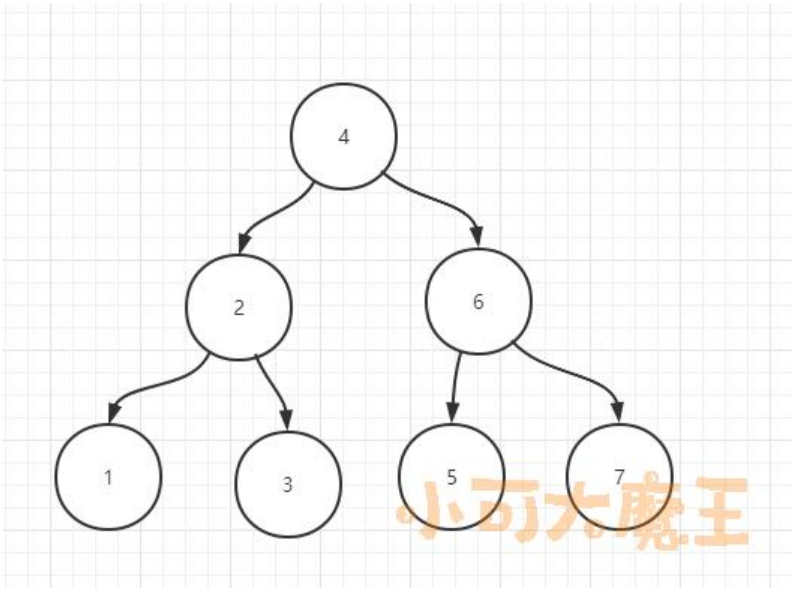
一. 了解索引树(B+树)

mysql的索引是由B+树实现的,在研究索引之前,先了解1下B+树的进化之路

1. 二叉查找树

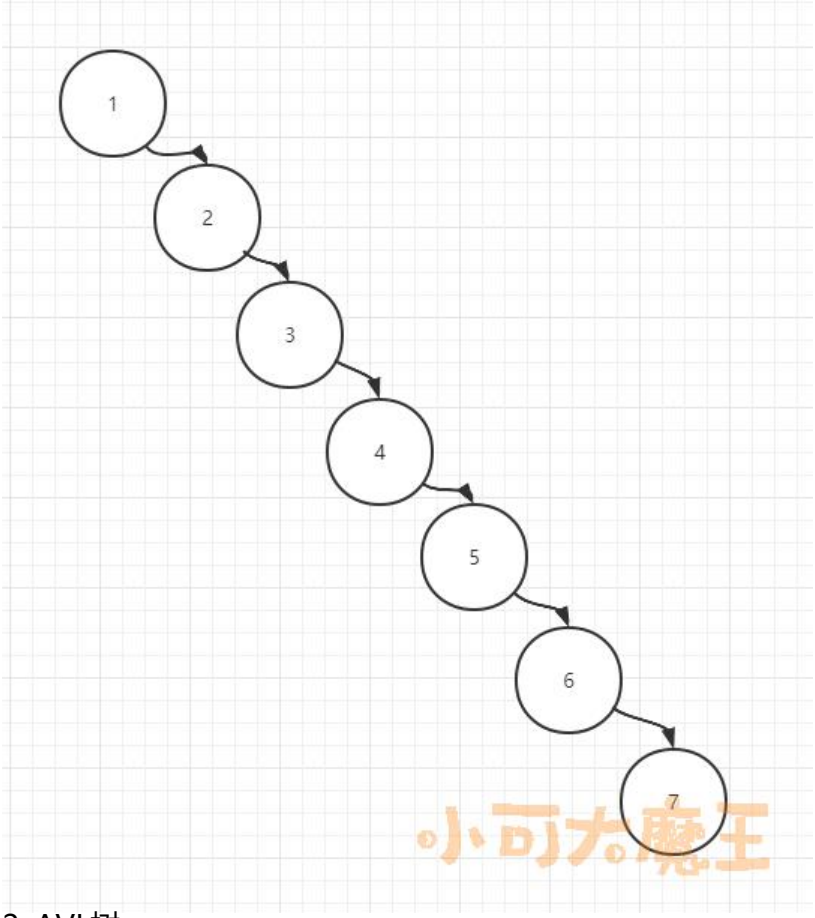
二叉查找树是有序的树

如下图:(理想状态)



缺点:

在插入的时候容易变成如下形状



2. AVL树

自平衡二叉查找树

带有平衡条件的二叉查找树, 通过左旋和右旋, 会强制维护树变成**图一**的想状态, 高度差不会大于1,所以随便插入几条数据, 就要进行左旋右旋调整

缺点: 插入效率低下, 并且每个节点只有2个子节点, 数据量上去之后树的高度过高, 高度高了代表IO次的增加

3. 红黑树,SB树

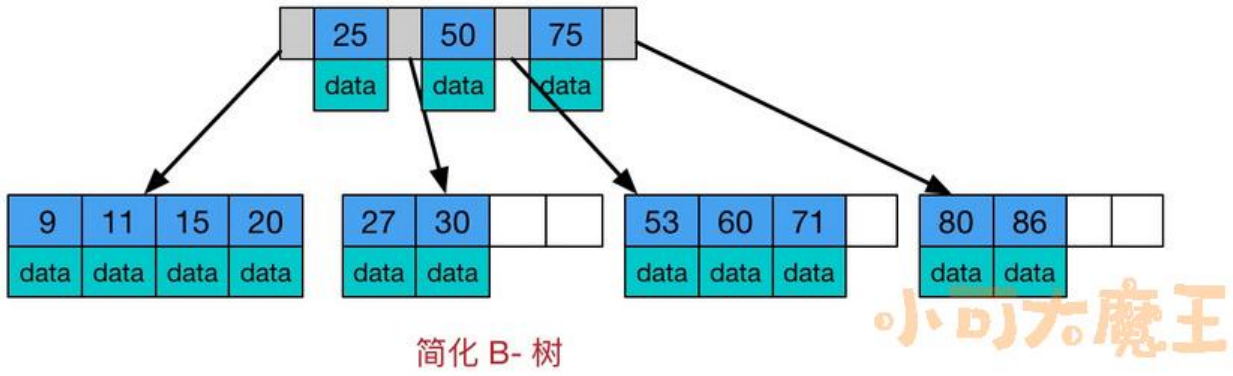
再次优化了平衡性,没有那么高的平衡性要求, 但是仍然没有解决子节点数量的问题, 一般这2种树在内中使用

4. B-树(B树,不是B减树)

优化了子节点数量

缺点: 所有键值分布在整颗树中**（索引值和具体data都在每个节点里）** ,mysql默认每个节点16K, 设每条数据1K, 每个节点只能存6条数据, 所以百万数据就能让树达到9层 想找到对应的数据, 磁盘IO数过多, 而且范围查询只能通过中序遍历来定位最小值和最大值

如下图:(网上找的,自己画累死了)

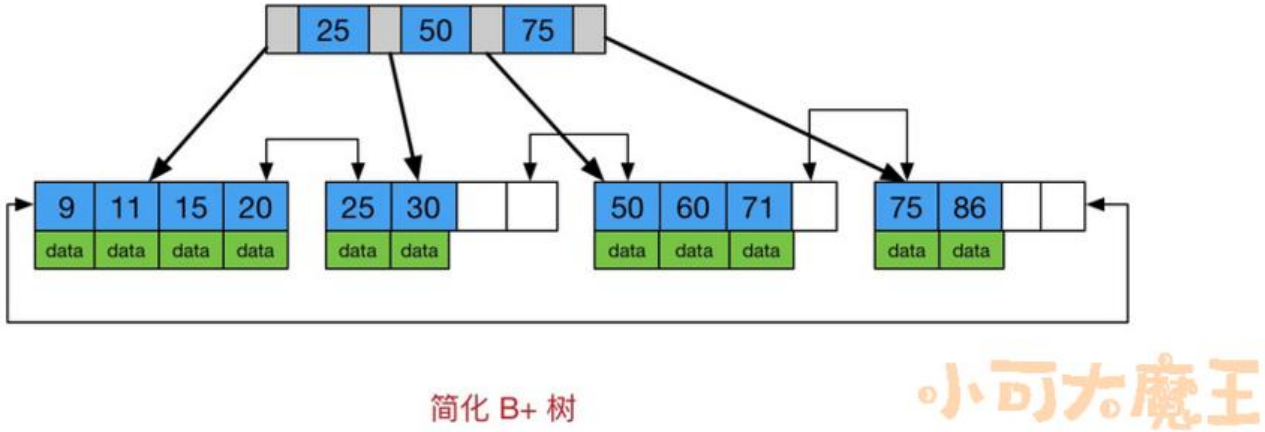


5. B+树

优化了数据只放在叶子节点, 非叶子节点只是索引值, 并且叶子节点有指针指向下一个节点

mysql每个索引节点16K **(默认,可以设置)** , 如果只是1个int4个字节的索引 + 6个字节的指针, 每个节能存1638个索引数据, (假设每条数据1K) 所以3层B+树就能包含1638 * 1638 * 4=268万条数据,当数量没有超过268万的时候,只要经过3次IO就能找到数据

如下图: (网上找的,自己画累死了)



二. 聚簇索引 和 非聚簇索引

1. 聚簇索引顺序和物理存储顺序是一致的

innodb默认主键是聚簇索引, 如果没有主键, mysql会选择非空的唯一索引当主键,并且用作聚簇索引, 多个非空的唯一索引会优先选择第一个, 如果也没有非空的唯一索引, 会生成1个隐藏的列,DB_ROW_ID当主键

总结: 主键一定是聚簇索引, 聚簇索引不一定是主键

2. 非聚簇索引的顺序和物理存储无关, 非聚簇索引叶子节点存储的是聚簇索引的值, 然后 **回表**再去聚索引查出真正的记录在哪

三. 覆盖索引

原文链接：mysql 的索引

● 覆盖索引就是select **查询的字段** from table where xxx 其中查询那些字段 在索引中已经存在了, 所以不需要回表去找到真正的记录

举例:

创建组合索引 age,salary,name

现在你想要找到所有年龄大于等于30岁,工资小于10000的员工

select age , salary , name from emp where age >= 30 and salary < 10000

由于在索引中已经包含了age , salary , name 的值, 所以无须回表

四. 哪些情况索引会失效

1. is null 和 is not null 和 !=, <> 不会走索引
2. 隐式类型转换, 比如字符串不加引号
3. 索引列做计算或者使用函数 比如: select * from user where id - 1 = 8
4. like 百分号开头, 比如 select * from user where name like "%XX"
5. 组合索引不满足最左匹配原则,

比如 索引是**(gender, age,salary)** 查询语句如下

```
select * from age= 20 and salary = 20000 (不走索引)
select * from gender = 1 and salary = 20000 (只走了gender的索引)
```

6. 组合索引前面的列使用范围查询 >, <, like, 后面的字段不走索引, 但是 >=, <= 可以

比如 索引是**(gender, age,salary)** 查询语句如下

```
select * from gender >= 0 and age = 20 and salary = 20000 (走索引)
select * from gender > 0 and age = 20 and salary = 20000 (只走了gender的索引)
select * from gender >= 0 and age >= 20 and salary > 20000 (全走)
```

7. 关联表的时候,关联字段长度不一样,比如都是name字段,A表name 是varchar(20),B表name 是varchar(22), 编码不一致也会导致失效
8. 使用or,除非or的字段都是索引, 不然会导致索引失效
9. 执行器误判, 执行器有概率误判, 觉得走索引比全表扫描还慢, 此时可以强制走索引

```
select * from table force index(PRI);(强制使用主键)

select * from table force index(indexName);(强制使用索引"indexName")
```

五. 索引的类型

1. 主键索引
2. 唯一索引
3. 普通索引
4. 组合索引(可以是唯一, 也可以是普通)
5. 全文索引(从来没用过, 有需要的自己去研究)

6. hash索引

在mysql中, 只有memory才支持hash索引, 只能支持 = , in , <=> **(和<>, != 不一样)**

7. 空间索引**(R-Tree)**

myisam支持空间索引，可以用作地理数据存储，R-Tree无须前缀索引。空间索引会从所有维度来索数据。查询时，可以有效地使用任意维度来组合查询。(完全没用过,随便网上抄的)

六. 索引下推

尽可能的过滤不符合条件的记录，哪怕不符合最左匹配原则

简单来说: 早期版本, 组合索引只使用了部分字段,另外部分字段由于各种原因没有用到(比如破坏了最匹配原则), 但是where条件有, 会把记录查出来之后再过滤, 同样情况下, 新版本(5.6)会先过滤了再回查出数据

举例:

组合索引(salary,name)

查询语句: select * from user where salary > 20000 and name like "%张%"

这条语句虽然只用到了salary索引字段, 在mysql5.6之前, 会把所有salary 大于 20000的记录主键拿出,回表查出记录, 再去判断是否满足like "%张%", 在mysql5.6后, 因为索引中有name字段, 所以在索引先判断是否满足like "%张%" ,如果满足才把主键拿去回表查, 少了很多回表操作, 回表频繁会产生大量IO