目录

[一. 索引 1](#_Toc1548435384)

[1. 索引的定义 1](#_Toc1346664542)

[2. 二分查找 1](#_Toc1060801661)

[3.二叉树(Binary Tree) 2](#_Toc484279033)

[1.二叉树 2](#_Toc314685501)

[2.平衡二叉树 2](#_Toc1814476393)

[4.B树/B+树 3](#_Toc1636949751)

[1.B树的定义 4](#_Toc801463340)

[2.B+树的定义： 4](#_Toc1176921396)

[3.B+树的作用 4](#_Toc46030055)

[4.B+树的操作 5](#_Toc533021465)

[5.B+树的扇出(fan out) 6](#_Toc1337470618)

[6.B+树存储数据举例 6](#_Toc1157343577)

[7.B和B+树的区别 9](#_Toc1714107760)

[8.MyISAM与InnoDB索引实现方式对比 10](#_Toc515997815)

[二.explain解读 12](#_Toc836310119)

[1.示例 12](#_Toc583700418)

[2.列解读 12](#_Toc547625830)

[3.案例优化分析 14](#_Toc1979897415)

[4.索引基数算法与合理性 15](#_Toc876743640)

**索引与explain**

# 一. 索引

## 1. 索引的定义

索引是对记录按照一个或者多个字段进行排序的一种方式。对表中的某个字段建立索引会创建另一种数据结构，其中保存着字段的值，每个值又指向与它相关的记录。这种索引的数据结构是经过排序的，因而可以对其执行二分查找。且性能较高。

## 2. 二分查找

二分查找法(binary search)也称为折半查找法，用来查找一组有序的记录数组中的某一记录，基本思想是:将记录有序化(递增或递减)排列，在查找过程中采用跳跃式方式查找，即先以有序数列的中点位置作为比较对象，如果要找的元素值小于该中点元素，则将待查序列缩小为左半部分，否则为右半部分。通过一次比较，将查找区间缩小一半。(O(logN)的时间复杂度)

给出一个例子，注意该例子已经是升序排序的，且查找数字 48

数据: 5， 10， 19， 21， 31， 37， 42， 48， 50， 52

下标: 0， 1， 2， 3， 4， 5， 6， 7， 8， 9

步骤一:设 low 为下标最小值0 ，high 为下标最大值 9

步骤二:通过 low和 high得到mid ，mid=(low + high) / 2，初始时mid 为下标 4 (也可以=5看具体算法实现);

步骤三 : mid=4 对应的数据值是31，31 < 48(我们要找的数字)

步骤四:通过二分查找,将low设置为31对应的下标4 ,high 保持不变为9 ,此时 mid 为 6

步骤五 : mid=6 对应的数据值是42, 42 < 48(我们要找的数字);

步骤六:通过二分查找,将low设置为42对应的下标 6 ,high 保持不变为 9 ,此时 mid 为 7

步骤七 : mid=7 对应的数据值是48，48 == 48(我们要找的数字)，查找结束。

通过3次二分查找就找到了我们所要的数字,而顺序查找 则需要8次。

## 3.二叉树(Binary Tree)

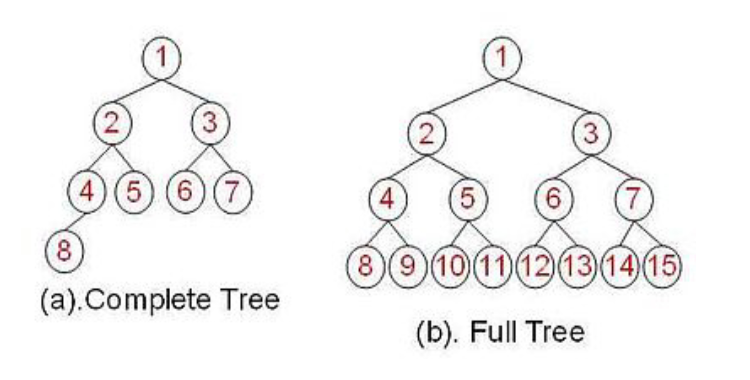
### 1.二叉树

• 每个节点至多只有二棵子树;

• 二叉树的子树有左右之分，次序不能颠倒;

• 一棵深度为k，且有2k-1个节点，称为满二叉树(Full Tree);

• 一棵深度为k，且root到k-1层的节点树都达到最大，第k层的所有节点都连续集中在最左边，此时为完全二叉树(CompleteTree)

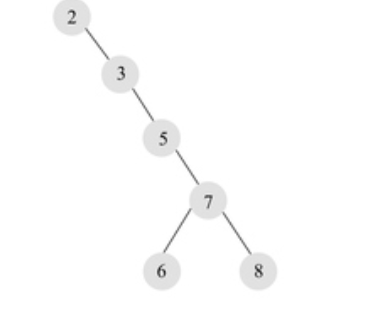


2.平衡二叉树

• 左子树和右子树都是平衡二叉树;

• 左子树和右子树的高度差绝对值不超过1;

平衡二叉树 非平衡二叉树

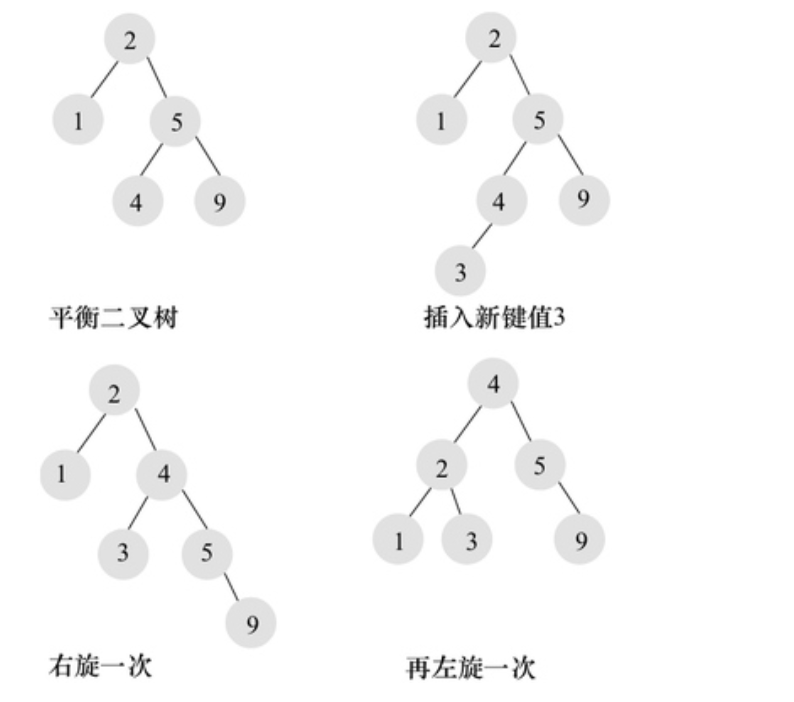
平衡二叉树的遍历，以上面平衡二叉树的图例为样本，进行遍历:

◦ 前序: 6,3,2,5,7,8 (ROOT节点在开头, 中-左-右 顺序)

◦ 中序: 2,3,5,6,7,8 (中序遍历即为升序，左-中-右 顺序)

◦ 后序: 2,5,3,8,7,6 (ROOT节点在结尾，左-右-中 顺序)

平衡二叉树的旋转



总结：通过旋转(左旋/右旋)来维护平衡二叉树的平衡，在添加和删除的时候需要有额外的开销。

## 4.B树/B+树

（注意:B树和B+树开头B不是Binary，而是 Balance，即平衡多路查找树。）

### 1.B树的定义

阶为M的B树的定义:

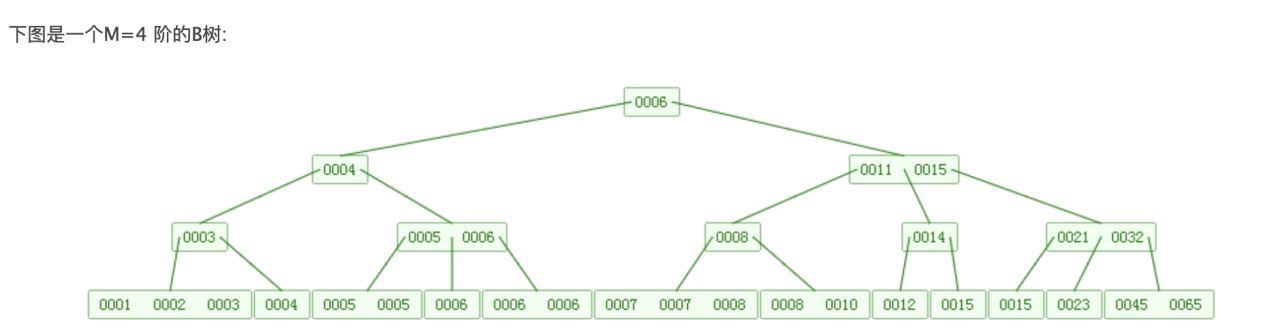
• 每个节点最多有M个孩子

• 除了root节点外，每个非叶子(non-leaf)节点至少含有(M/2)个孩子;

• 如果root节点不为空，则root节点至少要有两个子节点;

• 一个非叶子(non-leaf)节点如果含有K个孩子，则包含k-1个keys;

• B树中的非叶子(non-leaf)节点也包含了数据部分;



### 2.B+树的定义：

• 数据只存储在叶子节点上，非叶子节点只保存索引信息;

• 非叶子节点(索引节点)存储的只是一个Flag，不保存实际数据记录;

• 索引节点指示该节点的左子树比这个Flag小，而右子树大于等于这个Flag

• 叶子节点本身按照数据的升序排序进行链接(串联起来);

• 叶子节点中的数据在物理存储上是无序的，仅仅是在逻辑上有序(通过指针串联);



### 3.B+树的作用

• 在块设备上，通过B+树可以有效的存储数据;

• 所有记录都存储在叶子节点上，非叶子节点(non-leaf)存储索引(keys)信息;

• B+树含有非常高的扇出(fanout)，通常超过100，在查找一个记录时，可以有效的减少IO操作;

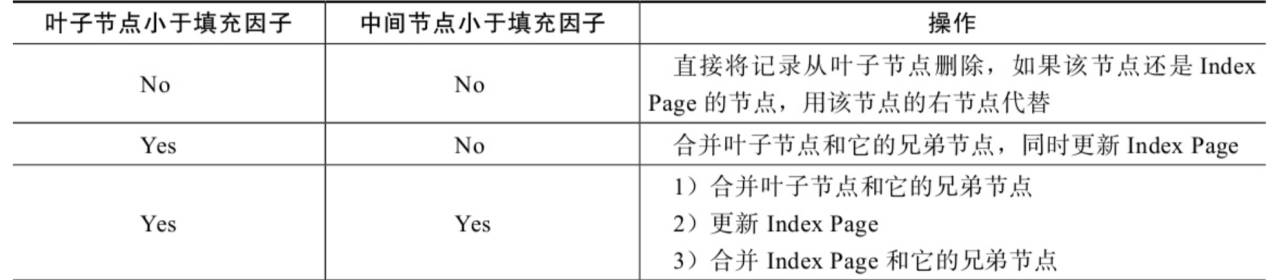
### 4.B+树的操作

• B+树的插入必须保证插入后叶子节点中的记录依然排序。



• B+树的删除

B+树使用填充因子(fill factor)来控制树的删除变化，50%是填充因子可设的最小值。B+树的删除操作同样必须保证删除后叶子节点中的记录依然排序。与插入不同的是，删除根据填充因子的变化来衡量。



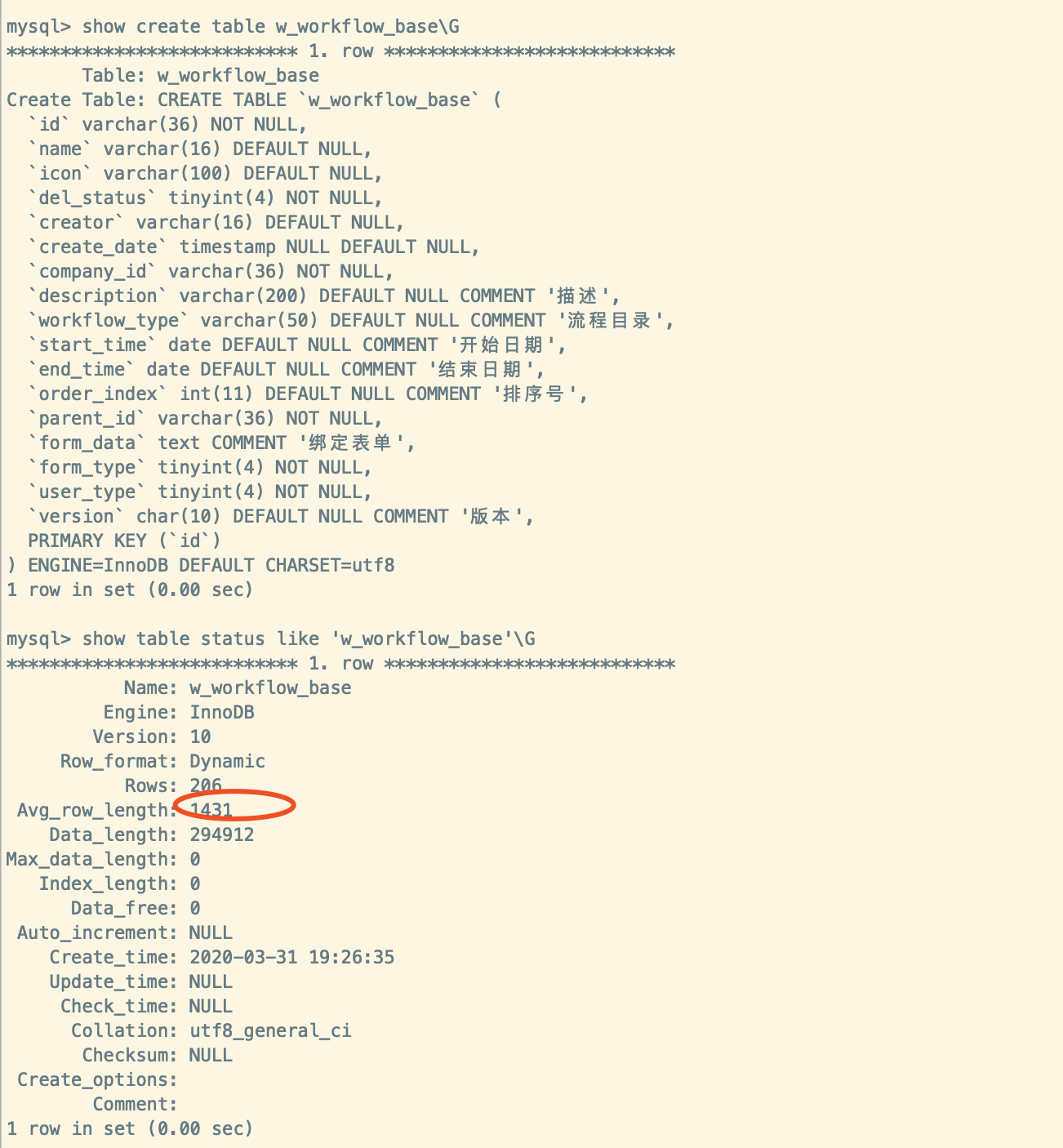
### 5.B+树的扇出(fan out)

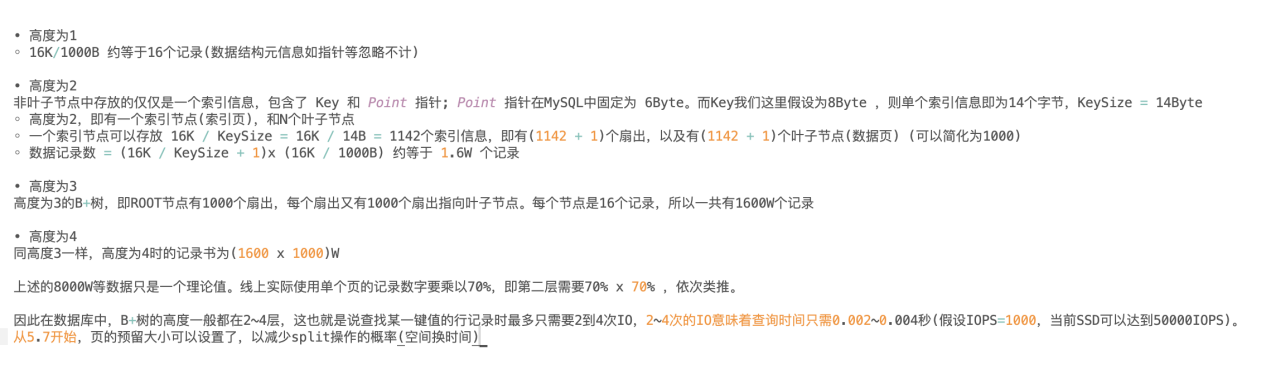


### 6.B+树存储数据举例

假设B+树中页的大小是16K，每行记录是1000Byte大小，求出树的高度为1，2，3，4时，分别可以存储多少条记录。







• 高度为1

◦ 16K/1000B 约等于16个记录(数据结构元信息如指针等忽略不计)

• 高度为2

非叶子节点中存放的仅仅是一个索引信息，包含了 Key 和 Point 指针; Point 指针在MySQL中固定为 6Byte。而Key我们这里假设为8Byte ，则单个索引信息即为14个字节， KeySize = 14Byte

◦ 高度为2，即有一个索引节点(索引页)，和N个叶子节点

◦ 一个索引节点可以存放 16K / KeySize = 16K / 14B = 1142个索引信息，即有(1142 + 1)个扇出，以及有(1142 + 1)个叶子节点(数据页) (可以简化为1000)

◦ 数据记录数 = (16K / KeySize + 1)x (16K / 1000B) 约等于 1.6W 个记录

• 高度为3

高度为3的B+树，即ROOT节点有1000个扇出，每个扇出又有1000个扇出指向叶子节点。每个节点是16个记录，所以一共有 1600W个记录

• 高度为4

同高度3一样，高度为4时的记录书为(1600 x 1000)W

上述的8000W等数据只是一个理论值。线上实际使用单个页的记录数字要乘以70%，即第二层需要70% x 70% ，依次类推。 因此在数据库中，B+树的高度一般都在2~4层，这也就是说查找某一键值的行记录时最多只需要2到4次IO，2~4次的IO意味着查询时间只需0.002~0.004秒(假设IOPS=1000，当前SSD可以达到50000IOPS)。

从MySQL5.7开始，页的预留大小可以设置了，以减少split操作的概率(空间换时间)

### 7.B和B+树的区别

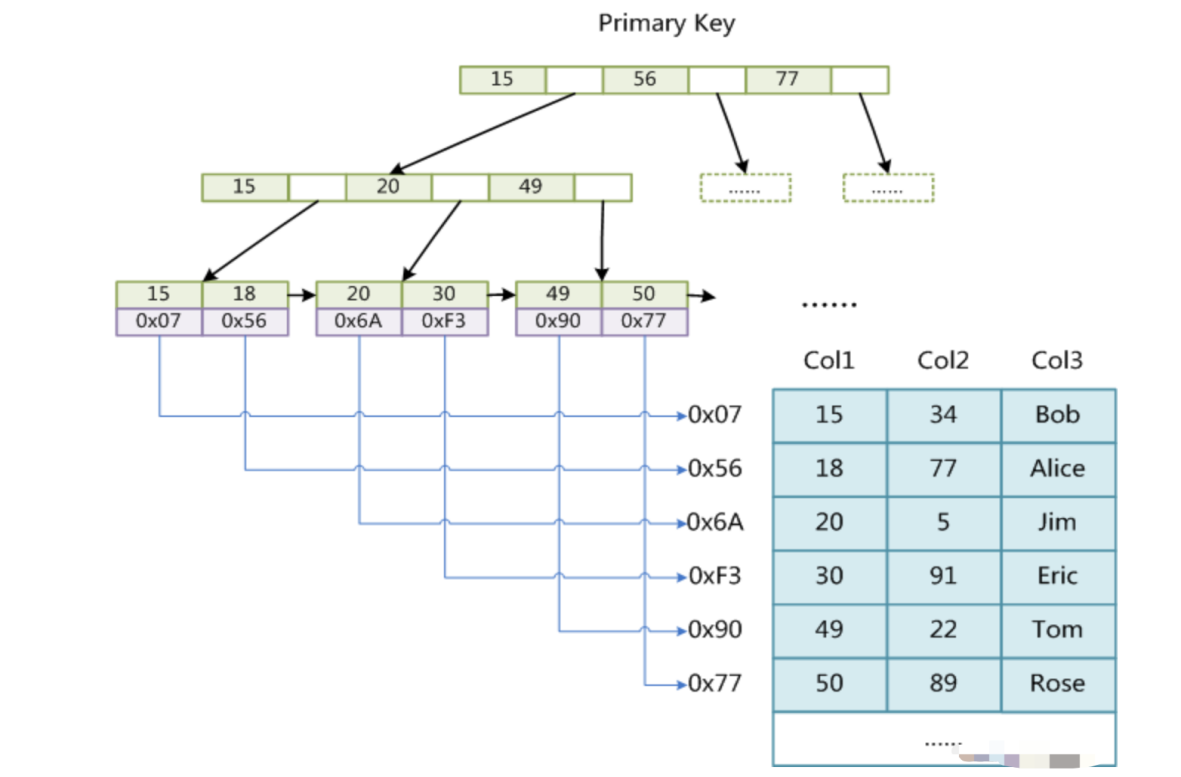
B+树的非叶子结点只包含索引信息，不包含实际value，所有的叶子节点和相连的节点使用链表相连，便于区间查找和遍历。

B+树的优点在于由于B+树在内部节点上不包含数据信息，因此在内存页中能够存放更多的key。 数据存放的更加紧密，具有更好的空间局部性。因此访问叶子节点上关联的数据也具有更好的缓存命中率。

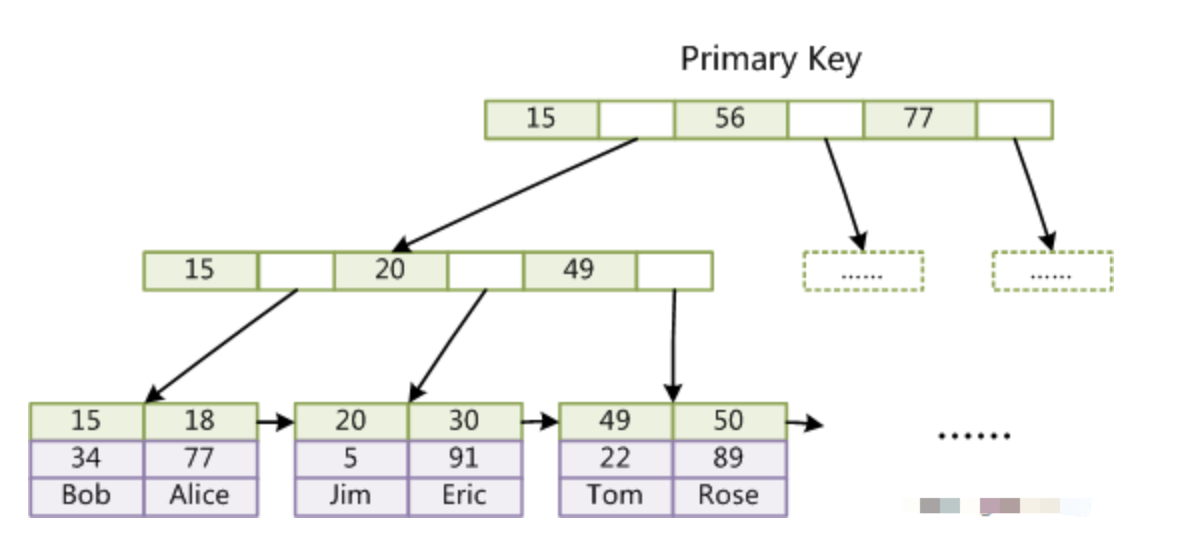
B+树的叶子结点都是相连的，因此对整棵树的遍历只需要一次线性遍历叶子节点即可。而且由于数据顺序排列并且相连，所以便于区间查找和搜索。而B树则需要进行每一层的递归遍历。相邻的元素可能在内存中不相邻，所以缓存命中性没有B+树好。

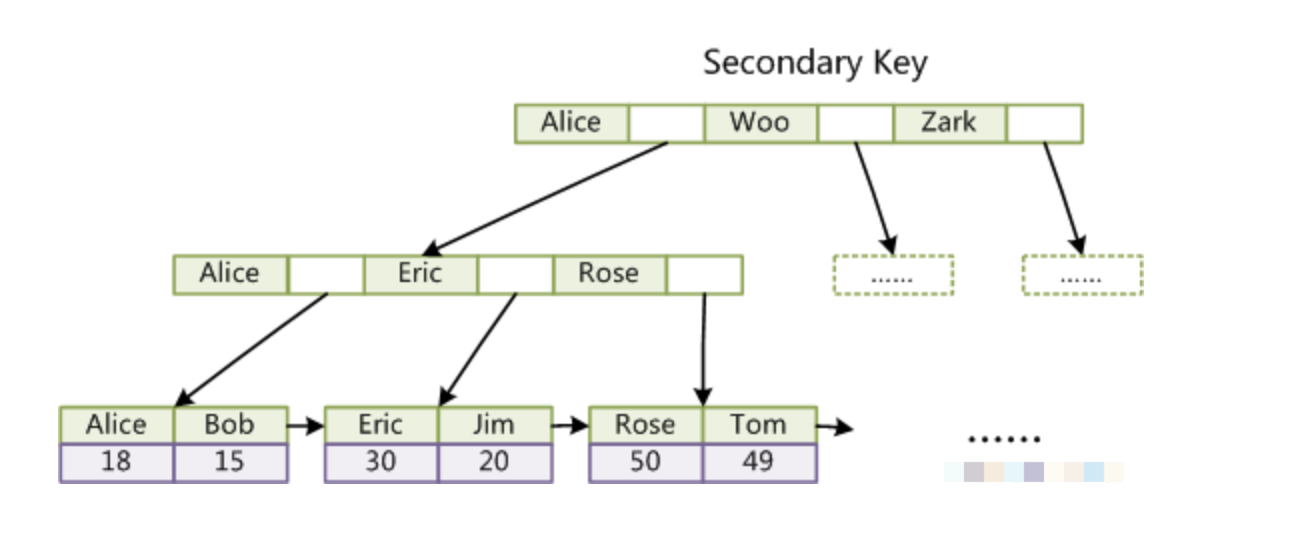
但是B树也有优点，其优点在于，由于B树的每一个节点都包含key和value，因此经常访问的元素可能离根节点更近，因此访问也更迅速。

### 8.MyISAM与InnoDB索引实现方式对比









9.索引实现方式差异的结论：

• 了解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。

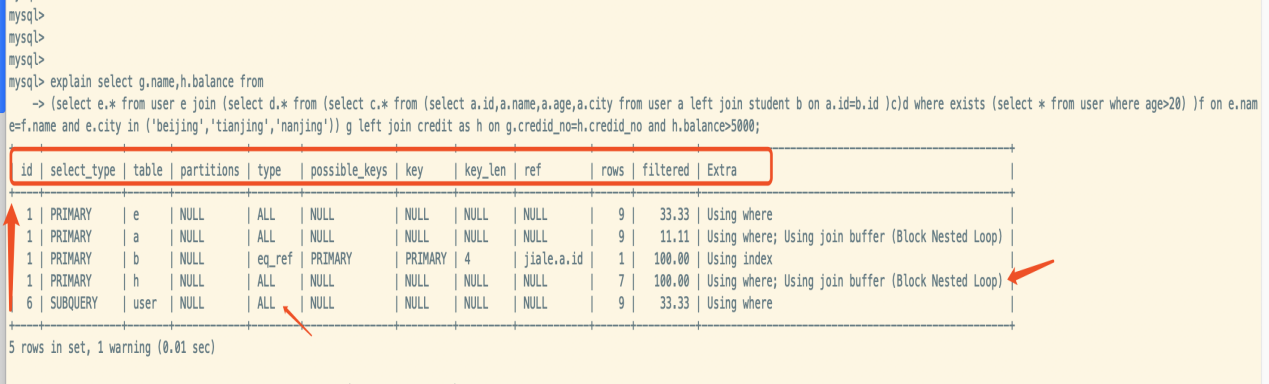
• 例如了解InnoDB的索引实现后，就很容易明白为什么不建议使用过长的字段作为主键，因为所有辅助索引都引用主索引，过长的主索引会使辅助索引变得过大。

• 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。

# 二.explain解读

## 1.示例



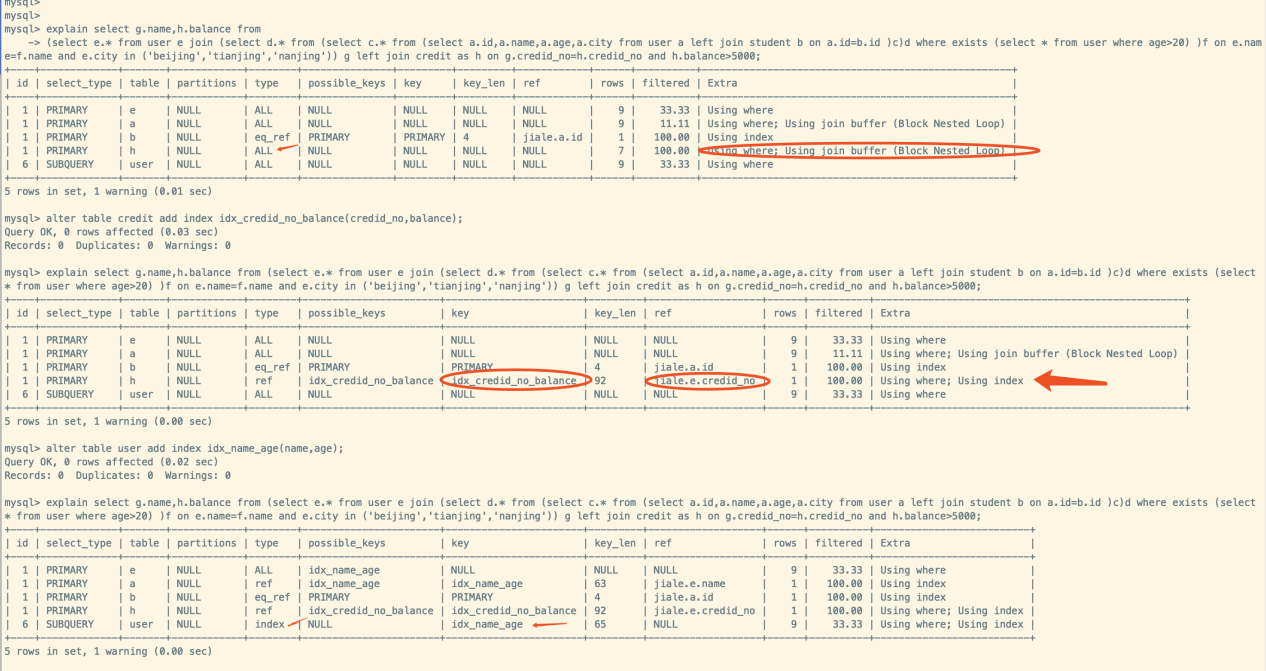
## 2.列解读

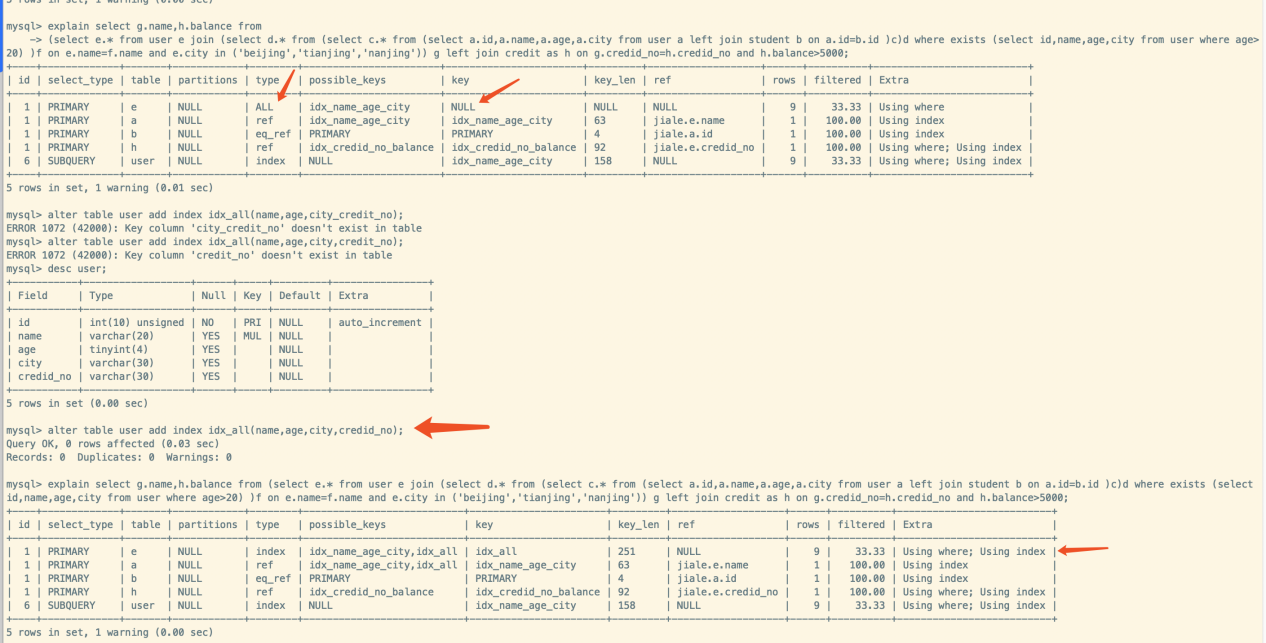






## 3.案例优化分析





mysql> explain format=json select g.name,h.balance from (select e.\* from user e join (select d.\* from (select c.\* from (select a.id,a.name,a.age,a.city from user a left join student b on a.id=b.id )c)d where exists (select \* from user where age>20) )f on e.name=f.name and e.city in ('beijing','tianjing','nanjing')) g left join credit as h on g.credid\_no=h.credid\_no and h.balance>5000\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

EXPLAIN: {

"query\_block": {

"select\_id": 1,

"cost\_info": {

"query\_cost": "13.85"

},

"nested\_loop": [

{

"table": {

"table\_name": "e",

"access\_type": "index",

"possible\_keys": [

"idx\_name\_age\_city",

"idx\_all"

],

"key": "idx\_all",

"used\_key\_parts": [

"name",

"age",

"city",

"credid\_no"

],

"key\_length": "251",

"rows\_examined\_per\_scan": 9,

"rows\_produced\_per\_join": 3,

"filtered": "33.33",

"using\_index": true,

"cost\_info": {

"read\_cost": "2.20",

"eval\_cost": "0.60",

"prefix\_cost": "2.80",

"data\_read\_per\_join": "768"

},

"used\_columns": [

"name",

"city",

"credid\_no"

],

"attached\_condition": "((`jiale`.`e`.`city` in ('beijing','tianjing','nanjing')) and (`jiale`.`e`.`name` is not null))"

}

},

{

"table": {

"table\_name": "a",

"access\_type": "ref",

"possible\_keys": [

"idx\_name\_age\_city",

"idx\_all"

],

"key": "idx\_name\_age\_city",

"used\_key\_parts": [

"name"

],

"key\_length": "63",

"ref": [

"jiale.e.name"

],

"rows\_examined\_per\_scan": 1,

"rows\_produced\_per\_join": 3,

"filtered": "100.00",

"using\_index": true,

"cost\_info": {

"read\_cost": "3.00",

"eval\_cost": "0.60",

"prefix\_cost": "6.40",

"data\_read\_per\_join": "768"

},

"used\_columns": [

"id",

"name"

]

}

},

{

"table": {

"table\_name": "b",

"access\_type": "eq\_ref",

"possible\_keys": [

"PRIMARY"

],

"key": "PRIMARY",

"used\_key\_parts": [

"id"

],

"key\_length": "4",

"ref": [

"jiale.a.id"

],

"rows\_examined\_per\_scan": 1,

"rows\_produced\_per\_join": 3,

"filtered": "100.00",

"using\_index": true,

"cost\_info": {

"read\_cost": "3.00",

"eval\_cost": "0.60",

"prefix\_cost": "10.00",

"data\_read\_per\_join": "768"

},

"used\_columns": [

"id"

]

}

},

{

"table": {

"table\_name": "h",

"access\_type": "ref",

"possible\_keys": [

"idx\_credid\_no\_balance"

],

"key": "idx\_credid\_no\_balance",

"used\_key\_parts": [

"credid\_no"

],

"key\_length": "92",

"ref": [

"jiale.e.credid\_no"

],

"rows\_examined\_per\_scan": 1,

"rows\_produced\_per\_join": 4,

"filtered": "100.00",

"using\_index": true,

"cost\_info": {

"read\_cost": "3.02",

"eval\_cost": "0.84",

"prefix\_cost": "13.86",

"data\_read\_per\_join": "470"

},

"used\_columns": [

"credid\_no",

"balance"

],

"attached\_condition": "<if>(is\_not\_null\_compl(h), (`jiale`.`h`.`balance` > 5000), true)"

}

}

],

"optimized\_away\_subqueries": [

{

"dependent": false,

"cacheable": true,

"query\_block": {

"select\_id": 6,

"cost\_info": {

"query\_cost": "2.80"

},

"table": {

"table\_name": "user",

"access\_type": "index",

"key": "idx\_name\_age\_city",

"used\_key\_parts": [

"name",

"age",

"city"

],

"key\_length": "158",

"rows\_examined\_per\_scan": 9,

"rows\_produced\_per\_join": 2,

"filtered": "33.33",

"using\_index": true,

"cost\_info": {

"read\_cost": "2.20",

"eval\_cost": "0.60",

"prefix\_cost": "2.80",

"data\_read\_per\_join": "767"

},

"used\_columns": [

"age"

],

"attached\_condition": "(`jiale`.`user`.`age` > 20)"

}

}

}

]

}

}

1 row in set, 1 warning (0.00 sec)

mysql> explain format=json select g.name,h.balance from (select e.\* from user e join (select d.\* from (select c.\* from (select a.id,a.name,a.age,a.city from user a left join student b on a.id=b.id and a.age>20)c)d )f on e.name=f.name and e.city in ('beijing','tianjing','nanjing')) g left join credit as h on g.credid\_no=h.credid\_no and h.balance>5000\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

EXPLAIN: {

"query\_block": {

"select\_id": 1,

"cost\_info": {

"query\_cost": "13.85"

},

"nested\_loop": [

{

"table": {

"table\_name": "e",

"access\_type": "index",

"possible\_keys": [

"idx\_name\_age\_city",

"idx\_all"

],

"key": "idx\_all",

"used\_key\_parts": [

"name",

"age",

"city",

"credid\_no"

],

"key\_length": "251",

"rows\_examined\_per\_scan": 9,

"rows\_produced\_per\_join": 3,

"filtered": "33.33",

"using\_index": true,

"cost\_info": {

"read\_cost": "2.20",

"eval\_cost": "0.60",

"prefix\_cost": "2.80",

"data\_read\_per\_join": "768"

},

"used\_columns": [

"name",

"city",

"credid\_no"

],

"attached\_condition": "((`jiale`.`e`.`city` in ('beijing','tianjing','nanjing')) and (`jiale`.`e`.`name` is not null))"

}

},

{

"table": {

"table\_name": "a",

"access\_type": "ref",

"possible\_keys": [

"idx\_name\_age\_city",

"idx\_all"

],

"key": "idx\_name\_age\_city",

"used\_key\_parts": [

"name"

],

"key\_length": "63",

"ref": [

"jiale.e.name"

],

"rows\_examined\_per\_scan": 1,

"rows\_produced\_per\_join": 3,

"filtered": "100.00",

"using\_index": true,

"cost\_info": {

"read\_cost": "3.00",

"eval\_cost": "0.60",

"prefix\_cost": "6.40",

"data\_read\_per\_join": "768"

},

"used\_columns": [

"id",

"name",

"age"

]

}

},

{

"table": {

"table\_name": "b",

"access\_type": "eq\_ref",

"possible\_keys": [

"PRIMARY"

],

"key": "PRIMARY",

"used\_key\_parts": [

"id"

],

"key\_length": "4",

"ref": [

"jiale.a.id"

],

"rows\_examined\_per\_scan": 1,

"rows\_produced\_per\_join": 3,

"filtered": "100.00",

"using\_index": true,

"cost\_info": {

"read\_cost": "3.00",

"eval\_cost": "0.60",

"prefix\_cost": "10.00",

"data\_read\_per\_join": "768"

},

"used\_columns": [

"id"

],

"attached\_condition": "<if>(is\_not\_null\_compl(b), (`jiale`.`a`.`age` > 20), true)"

}

},

{

"table": {

"table\_name": "h",

"access\_type": "ref",

"possible\_keys": [

"idx\_credid\_no\_balance"

],

"key": "idx\_credid\_no\_balance",

"used\_key\_parts": [

"credid\_no"

],

"key\_length": "92",

"ref": [

"jiale.e.credid\_no"

],

"rows\_examined\_per\_scan": 1,

"rows\_produced\_per\_join": 4,

"filtered": "100.00",

"using\_index": true,

"cost\_info": {

"read\_cost": "3.02",

"eval\_cost": "0.84",

"prefix\_cost": "13.86",

"data\_read\_per\_join": "470"

},

"used\_columns": [

"credid\_no",

"balance"

],

"attached\_condition": "<if>(is\_not\_null\_compl(h), (`jiale`.`h`.`balance` > 5000), true)"

}

}

]

}

}

1 row in set, 1 warning (0.00 sec)

## 4.索引基数算法与合理性

MySQL5.6

SELECT

t.TABLE\_SCHEMA,

t.TABLE\_NAME,INDEX\_NAME,

s.CARDINALITY,

t.TABLE\_ROWS,

s.CARDINALITY/t.TABLE\_ROWS AS SELECTIVITY

FROM

information\_schema.TABLES t,

(SELECT database\_name,table\_name,index\_name,stat\_value AS CARDINALITY

FROM mysql.innodb\_index\_stats

WHERE (database\_name,table\_name,index\_name,stat\_name) IN

(SELECT table\_schema,table\_name,index\_name,CONCAT('n\_diff\_pfx0',MAX(seq\_in\_index))

FROM information\_schema.STATISTICS

-- where table\_name='xxxxx'

GROUP BY table\_schema , table\_name , index\_name )) s

WHERE t.table\_schema = s.database\_name

AND t.table\_name = s.table\_name

AND t.table\_rows != 0

AND t.table\_schema NOT IN ( 'mysql','performance\_schema','information\_schema')

ORDER BY SELECTIVITY;

MySQL5.7

SELECT

t.TABLE\_SCHEMA,t.TABLE\_NAME,INDEX\_NAME, CARDINALITY,

TABLE\_ROWS, CARDINALITY/TABLE\_ROWS AS SELECTIVITY

FROM

information\_schema.TABLES t,

(

SELECT table\_schema,table\_name,index\_name,cardinality

FROM information\_schema.STATISTICS

WHERE (table\_schema,table\_name,index\_name,seq\_in\_index) IN (

SELECT table\_schema,table\_name,index\_name,MAX(seq\_in\_index)

FROM information\_schema.STATISTICS

GROUP BY table\_schema , table\_name , index\_name )

) s

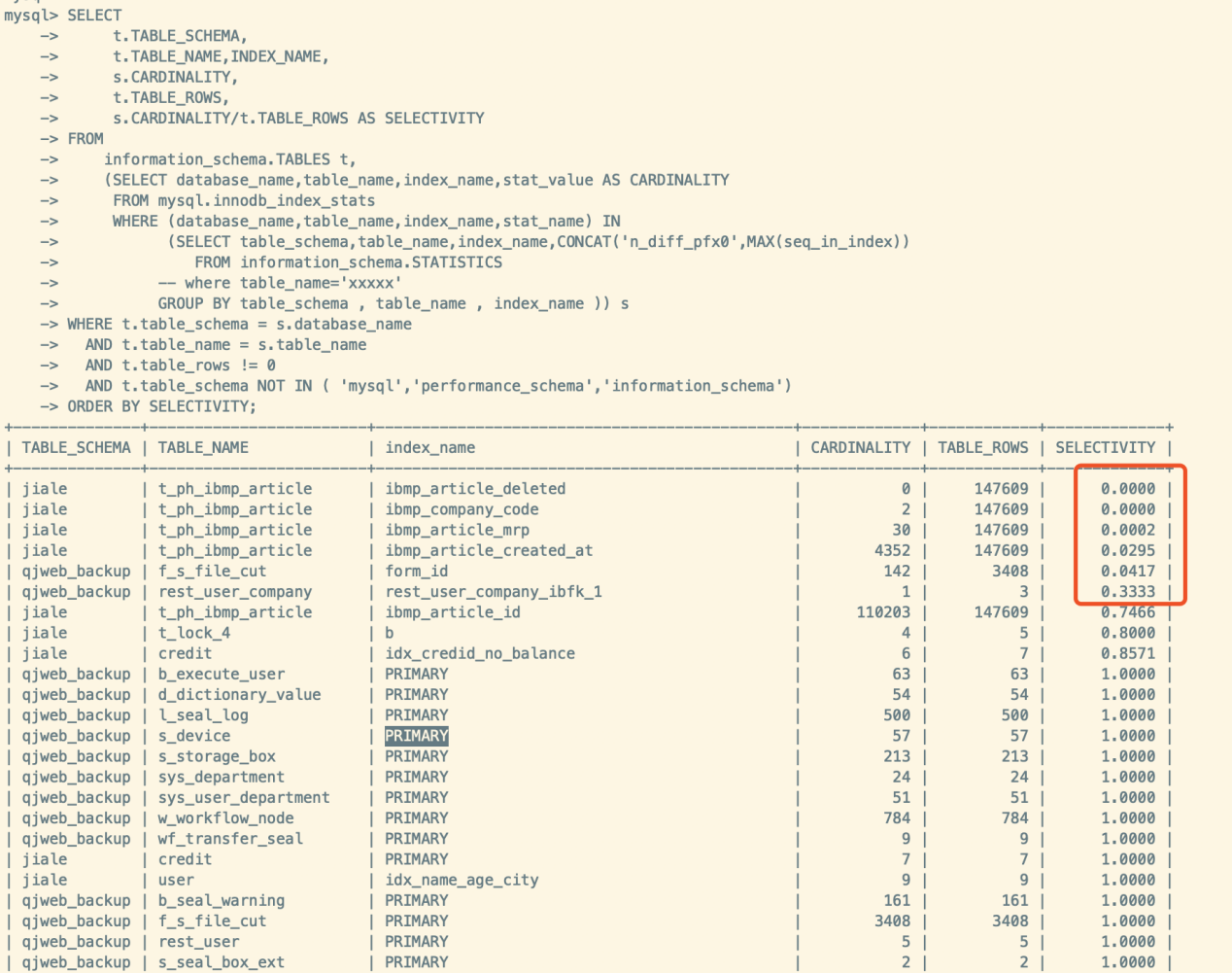
WHERE

t.table\_schema = s.table\_schema

AND t.table\_name = s.table\_name AND t.table\_rows != 0

AND t.table\_schema NOT IN ( 'mysql','performance\_schema','information\_schema')

ORDER BY SELECTIVITY;



DB-engine Rank

