mysql优化-------Myisam与innodb引擎,索引文件的区别

摘要： Myisam与innodb引擎,索引文件的区别： innodb的次索引指向对主键的引用。 myisam的次索引和主索引都指向物理行。 myisam一行一行的插入，会产生一行一行的文件，磁盘上有数据文件。

Myisam与innodb引擎,索引文件的区别：

innodb的次索引指向对主键的引用。

myisam的次索引和主索引都指向物理行。

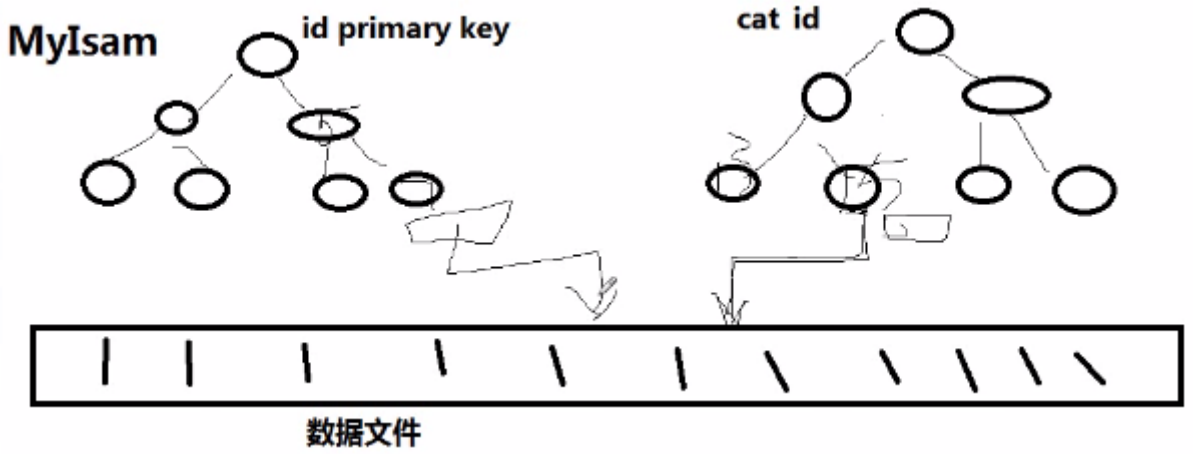
myisam一行一行的插入，会产生一行一行的文件，磁盘上有数据文件。

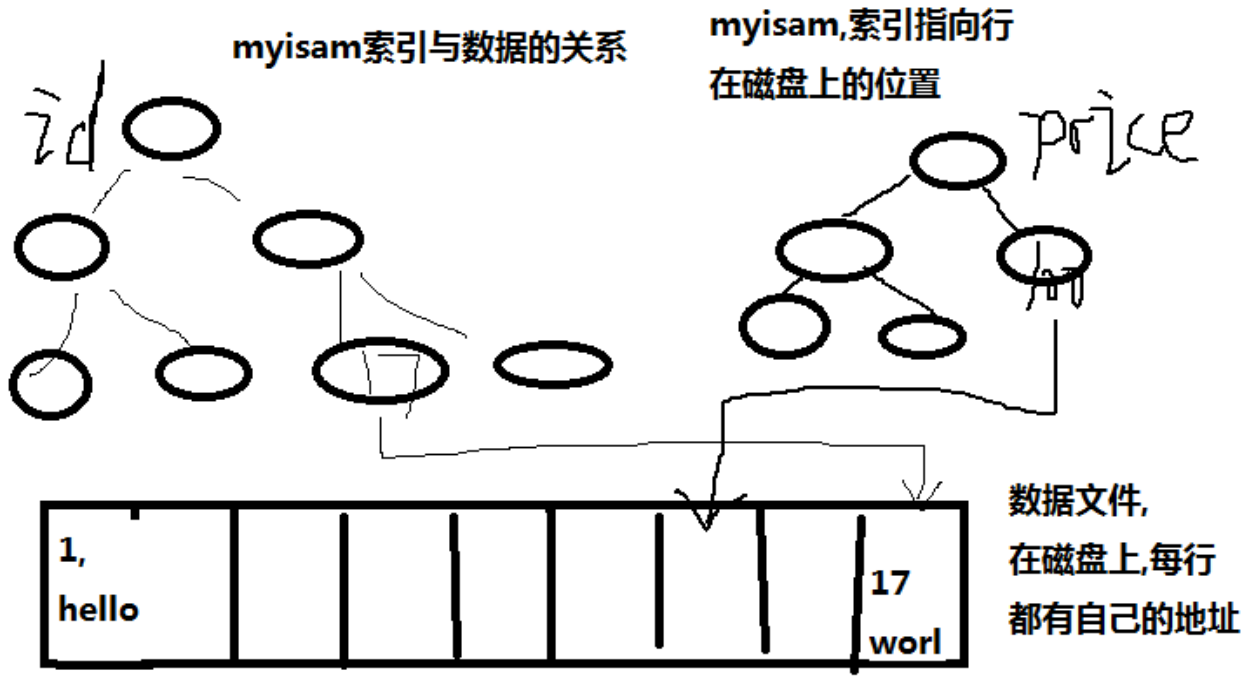
tree树的值是磁盘上物理位置的指针。

比如加了主键索引，索引排成一棵树的形状。首先根据id=7在主键索引的树上查找，查找到7之后就知道了7所在的物理行，然后就可以找到id=7的那一行数据了。

还有一个cat_id索引，根据cat_id=15可以找到数据所在的物理行。

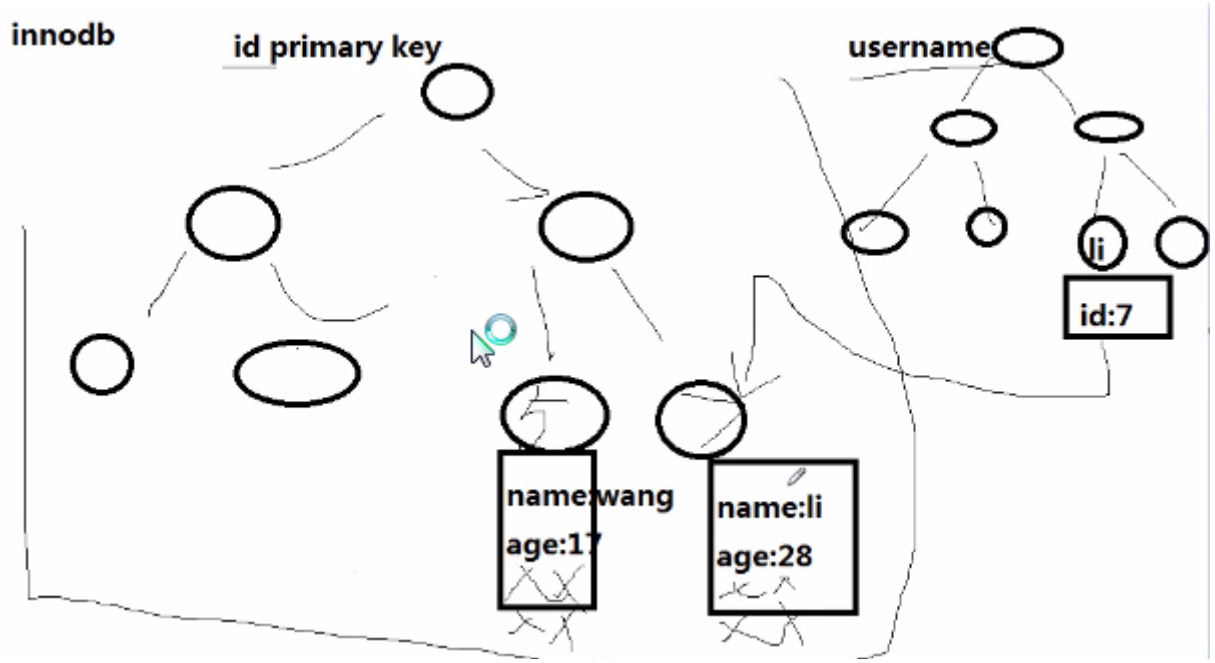
所以说myisam的次索引和主索引都指向物理行。



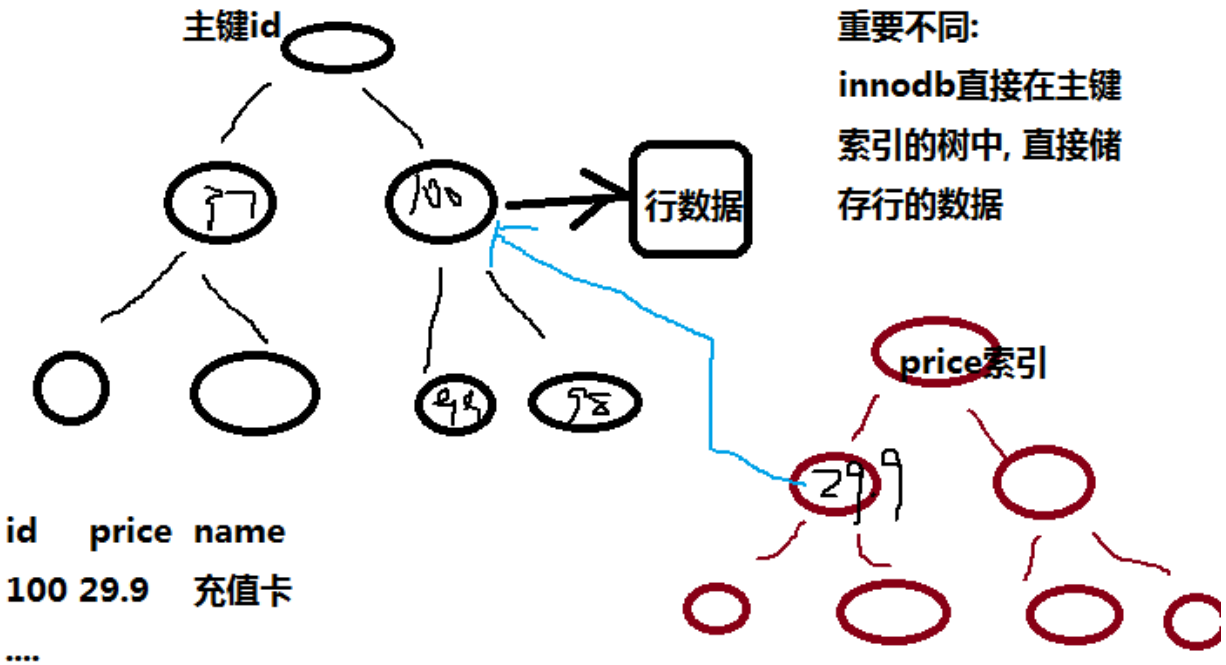


innodb的主键索引，数的每一个叶子下面，直接挂在了每行的数据，id=5的地方挂载的就是id=5的这行数据。数据就在叶子上，不用去磁盘上面查找。如果还有其他索引username，username=li的叶子下面放的是id=**7**.根据username索引这棵树上找到id=7然后在主键树上找到数据。

所以innodb的次索引指向对主键的引用。id的主索引成为聚簇索引，好处是根据主键查非常快，坏处是根据其他索引找的时候要多找一次主键这棵树。username是非聚簇索引。



innodb索引与数据的关系



innodb的主索引文件上 直接存放该行数据，称为聚簇索引，次索引指向对主键的引用。

myisam中， 主索引和次索引，都指向物理行（磁盘位置）。

注意： innodb来说，

1： 主键索引 既存储索引值，又在叶子中存储行的数据

2： 如果没有主键， 则会Unique key做主键

3： 如果没有unique，则系统生成一个内部的rowid做主键.

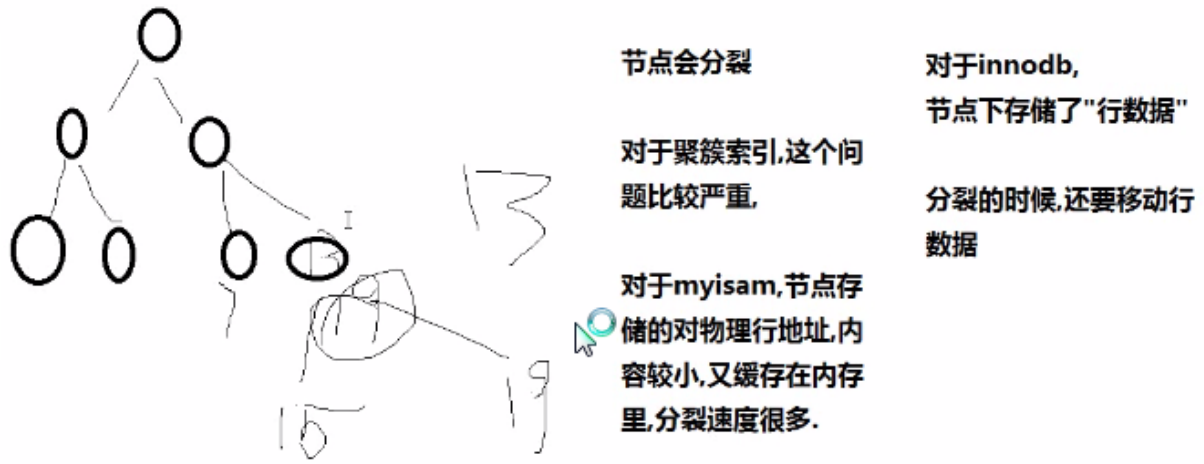
4： 像innodb中，主键的索引结构中，既存储了主键值，又存储了行数据，这种结构称为"聚簇索引"

聚簇索引

优势： 根据主键查询条目比较少时，不用回行（数据就在主键节点下）

劣势： 如果碰到不规则数据插入时，造成频繁的页分裂.

myisam中对于索引文件是要放在内存中缓存起来的。节点会分裂：原来19的节点后来来了15和13，则19的位置换成13，并在下面添加15，19。对于聚簇索引就很严重。对于myisam没什么，对于innodb就很麻烦。



高性能索引策略

对于innodb而言，因为节点下有数据文件，因此节点的分裂将会比较慢.

对于innodb的主键，尽量用整型，而且是递增的整型.

如果是无规律的数据，将会产生的页的分裂，影响速度.

```
create table A{
    id varchar(64) primary key,
    ver int,
}
```

在id和ver上有联合索引10000条数据。

为什么select id from A order by id很慢

而select id from A order by id,ver很快

如果用的是myisam，那么都用到了索引覆盖，应该是一样都很快，有可能不实用的myisam引擎。myisam无论使用什么索引都是指向物理行的位置。

如果是innodb引擎，每个叶子下面直接放的数据，这些数据比较大内存放不下，就放在磁盘上。innodb的主键是聚簇索引。有比较长的列，聚簇索引导致沿id排序时要跨好多块。而且块比较多。所以查找很慢。

第二句是联合索引，联合索引没有放数据块（除了主键索引其余索引都指向主键索引，不带数据），而是放的是主键索引的位置指向id的值，不带有数据，文件比较小可以在内存中存放。现在只是取出id不用回行，就是在索引文件中取，而且索引文件比较小就放在内存中，所以很快。第一个语句，也只是在索引文件中查找，发生了索引覆盖，但是这个主键索引文件比较大，而且不一定在内存中，查找主键树的时候来回跳跃就很慢。

如果把数据比较大的字段去掉，速度也会提升，因为查找主键索引文件来回跳的时候就不会慢了。

通过下面的规律可以看出-----

1： innodb的buffer_page 很强大.

2： 聚簇索引的主键值,应尽量是连续增长的值,而不是要是随机值,(不要用随机字符串或UUID)

否则会造成大量的页分裂与页移动.

实验： 聚簇索引使用随机值导致页频繁分裂影响速度

过程:建立innodb表， 利用php连接mysql，

分别规则插入10000条数据,不规则插入10000条数据

观察时间的差异，体会聚簇索引，页分裂的影响.

```
create table t5(
id int primary key,
c1 varchar(500),
c2 varchar(500),
c3 varchar(500),
c4 varchar(500),
c5 varchar(500),
c6 varchar(500)
) engine innodb charset utf8;

create table t6(
id int primary key,
c1 varchar(500),
c2 varchar(500),
c3 varchar(500),
c4 varchar(500),
c5 varchar(500),
c6 varchar(500)
) engine innodb charset utf8;
// testinnodb.php
$time_start = microtime_float();
$str = str_repeat('hello',100);
for($i=1;$i<=10000;$i++) {
   $sql = "insert into t5 values ($i,'$str' ,
'$str' , '$str' , '$str' , '$str' , '$str'
)";
```

```
    //echo $sql;
    mysql_query($sql , $conn);
}
$time_end = microtime_float();
echo 'seq insert cost' , ($time_end - $time_start)
, "seconds\n";
function microtime_float()
{
    list($usec, $sec) = explode(" ", microtime());
    return ((float)$usec + (float)$sec);
}
// rndinnodb.php
$base = range(1,10000);
shuffle($base);
$time_start = microtime_float();
$str = str_repeat('hello',100);
foreach($base as $i) {
    $sql = "insert into t6 values ($i,'$str' ,
'$str' , '$str' , '$str' , '$str' , '$str'
)";
    //echo $sql;
    mysql_query($sql , $conn);
}
$time_end = microtime_float();
echo 'rand insert cost' , ($time_end - $time_start)
, "seconds\n";
function microtime_float()
```

```
{
    list($usec, $sec) = explode(" ", microtime());
    return ((float)$usec + (float)$sec);
}
```

字段数	混乱程度(步长)	顺序1000条(秒数)	乱序1000条(秒数)	顺序写入page页数	乱序写入page数
1	1	54.365	53.438	62	91
10	1	53.413	62.940	235	1301
10	100		64.18		1329
10	1000		67.512		1325

通过上面的规律可以看出-----

1: innodb的buffer_page 很强大.

2: 聚簇索引的主键值,应尽量是连续增长的值,而不是要是随机值,(不要用随机字符串或UUID)

否则会造成大量的页分裂与页移动.