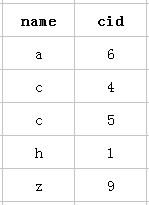
**MySQL最左前缀原则**

**个人总结:**

**最左原则 就是在复合索引中最左侧(第一个)字段最有用处;mysql会对这个字段最先排序，然后对第二字段排除。。。**

**那么问题来了，第一个已经排序了第二个怎么排?**

**那肯定在第一个基础上排了。比如**



**这个索引排序表中可以看出 name中有两个c那么第二个字段cid 就在这值得基础上排序，就是cid为4的排在cid为5的上面；**

**当然如果查询的时候不加第一个字段，其他字段的索引就会失效;**

**如 复合索引中（name,pwd,age）只查pwd和age -- “select \* from user where pwd =1 and age =1” 没有加name 就会失效**

**总结 :你建立的组合索引(a,b,c,d) 无论怎么使用只要和a没有组合(最左面的那个),那么都没有使用上索引!!!!!**

通过实例理解单列索引、多列索引以及最左前缀原则

实例：现在我们想查出满足以下条件的用户id：

mysql>SELECT ｀uid｀ FROM people WHERE lname｀=’Liu’  AND ｀fname｀=’Zhiqun’ AND ｀age｀=26

因为我们不想扫描整表，故考虑用索引。

单列索引：

ALTER TABLE people ADD INDEX lname (lname);

将lname列建索引，这样就把范围限制在lname=’Liu’的结果集1上，之后扫描结果集1，产生满足fname=’Zhiqun’的结果集2，再扫描结果集2，找到 age=26的结果集3，即最终结果。

由 于建立了lname列的索引，与执行表的完全扫描相比，效率提高了很多，但我们要求扫描的记录数量仍旧远远超过了实际所需 要的。虽然我们可以删除lname列上的索引，再创建fname或者age 列的索引，但是，不论在哪个列上创建索引搜索效率仍旧相似。

2.多列索引：

ALTER TABLE people ADD INDEX lname\_fname\_age (lame,fname,age);

为了提高搜索效率，我们需要考虑运用多列索引,由于索引文件以B－Tree格式保存，所以我们不用扫描任何记录，即可得到最终结果。

注：在mysql中执行查询时，只能使用一个索引，如果我们在lname,fname,age上分别建索引,执行查询时，只能使用一个索引，mysql会选择一个最严格(获得结果集记录数最少)的索引。

3.最左前缀：顾名思义，就是最左优先，上例中我们创建了lname\_fname\_age多列索引,相当于创建了(lname)单列索引，(lname,fname)组合索引以及(lname,fname,age)组合索引。

注：在创建多列索引时，要根据业务需求，where子句中使用最频繁的一列放在最左边。

拓展：在网上看到一个关于最左前缀原则提出这么一个例子。

多列字段做索引，state/city/zipCode，想要索引生效的话，只能使用如下的组合

state/city/zipCode

state/city

state

其他方式（如city，city/zipCode），则索引不会生效

这种现象是怎么导致的？和索引的存储方式有关吗？

本人页参考了下其他网友的观点，个人认为，所谓最左前缀原则就是先要看第一列，在第一列满足的条件下再看左边第二列，以此类推。有位网友描述得很形象：

你可以认为联合索引是闯关游戏的设计

例如你这个联合索引是state/city/zipCode

那么state就是第一关 city是第二关， zipCode就是第三关

你必须匹配了第一关，才能匹配第二关，匹配了第一关和第二关，才能匹配第三关

你不能直接到第二关的

索引的格式就是第一层是state，第二层才是city

索引是因为B+树结构  所以查找快  如果单看第三列  是非排序的。

多列索引是先按照第一列进行排序，然后在第一列排好序的基础上再对第二列排序，如果没有第一列的话，直接访问第二列，那第二列肯定是无序的，直接访问后面的列就用不到索引了

**很好的一个问题回答**

这是你的表结构，有三个字段，分别是id,name,cid

CREATE TABLE `student` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(255) DEFAULT NULL,

`cid` int(11) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `name\_cid\_INX` (`name`,`cid`),

) ENGINE=InnoDB AUTO\_INCREMENT=8 DEFAULT CHARSET=utf8

索引方面：id是主键，(name,cid)是一个多列索引。  
-----------------------------------------------------------------------------  
**下面是你有疑问的两个查询：**

EXPLAIN SELECT \* FROM student WHERE cid=1;

https://pic2.zhimg.com/80/d3086a6c81bb2c77796cfc2249b610bc_hd.jpg

EXPLAIN SELECT \* FROM student WHERE cid=1 AND name='小红';

https://pic3.zhimg.com/80/53ab2cdea64b7e58e66c4ef86aa6b06a_hd.jpg

**你的疑问是**：

sql查询用到索引的条件是必须要遵守最左前缀原则，为什么上面两个查询还能用到索引？

讲上面问题之前，我先补充一些知识，因为我觉得你对索引理解是狭隘的：  
上述你的两个查询的explain结果中显示用到索引的情况类型是不一样的。,可观察explain结果中的type字段。你的查询中分别是：  
1. type: index   
2. type: ref

**解释：**  
index：这种类型表示是mysql会对整个该索引进行扫描。要想用到这种类型的索引，对这个索引并无特别要求，只要是索引，或者某个复合索引的一部分，mysql都可能会采用index类型的方式扫描。但是呢，缺点是效率不高，mysql会从索引中的第一个数据一个个的查找到最后一个数据，直到找到符合判断条件的某个索引。

**所以：**对于你的第一条语句：

EXPLAIN SELECT \* FROM student WHERE cid=1;

判断条件是cid=1,而cid是(name,cid)复合索引的一部分，没有问题，可以进行index类型的索引扫描方式。explain显示结果使用到了索引，是index类型的方式。

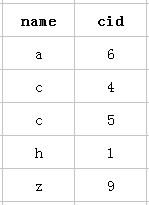
---------------------------------------------------------------------------------------------------------------------------

ref：这种类型表示mysql会根据特定的算法快速查找到某个符合条件的索引，而不是会对索引中每一个数据都进行一 一的扫描判断，也就是所谓你平常理解的使用索引查询会更快的取出数据。而要想实现这种查找，索引却是有要求的，要实现这种能快速查找的算法，索引就要满足特定的数据结构。**简单说，也就是索引字段的数据必须是有序的，才能实现这种类型的查找，才能利用到索引。**

有些了解的人可能会问，索引不都是一个有序排列的数据结构么。不过答案说的还不够完善，那只是针对单个索引，而复合索引的情况有些同学可能就不太了解了。

**下面就说下复合索引：**

以该表的(name,cid)复合索引为例,它内部结构简单说就是下面这样排列的：



mysql创建复合索引的规则是首先会对复合索引的最左边的，也就是第一个name字段的数据进行排序，在第一个字段的排序基础上，然后再对后面第二个的cid字段进行排序。其实就相当于实现了类似 order by name cid这样一种排序规则。

所以：第一个name字段是绝对有序的，而第二字段就是无序的了。所以通常情况下，直接使用第二个cid字段进行条件判断是用不到索引的，当然，可能会出现上面的使用index类型的索引。这就是所谓的mysql为什么要强调最左前缀原则的原因。

**那么什么时候才能用到呢?**

当然是cid字段的索引数据也是有序的情况下才能使用咯，什么时候才是有序的呢？观察可知，当然是在name字段是等值匹配的情况下，cid才是有序的。发现没有，观察两个name名字为 c 的cid字段是不是有序的呢。从上往下分别是4 5。

这也就是mysql索引规则中要求复合索引要想使用第二个索引，必须先使用第一个索引的原因。（而且第一个索引必须是等值匹配）。

---------------------------------------------------------------------------------------------------------------------------

所以对于你的这条sql查询：

EXPLAIN SELECT \* FROM student WHERE cid=1 AND name='小红';

没有错，而且复合索引中的两个索引字段都能很好的利用到了！因为语句中最左面的name字段进行了等值匹配，所以cid是有序的，也可以利用到索引了。

**你可能会问**：我建的索引是(name,cid)。而我查询的语句是cid=1 AND name='小红'; 我是先查询cid，再查询name的，不是先从最左面查的呀？

好吧，我再解释一下这个问题：首先可以肯定的是把条件判断反过来变成这样 name='小红' and cid=1; 最后所查询的结果是一样的。  
那么问题产生了？既然结果是一样的，到底以何种顺序的查询方式最好呢？

**所以**，而此时那就是我们的mysql查询优化器该登场了，mysql查询优化器会判断纠正这条sql语句该以什么样的顺序执行效率最高，最后才生成真正的执行计划。所以，当然是我们能尽量的利用到索引时的查询顺序效率最高咯，所以mysql查询优化器会最终以这种顺序进行查询执行。