5 索引（下）--优化索引

在下面的表中，如果执行select \* from T where k between 3 and 5 ,需要执行几次树的搜索操作，会扫描多少行？

create table T (

ID int primary key,

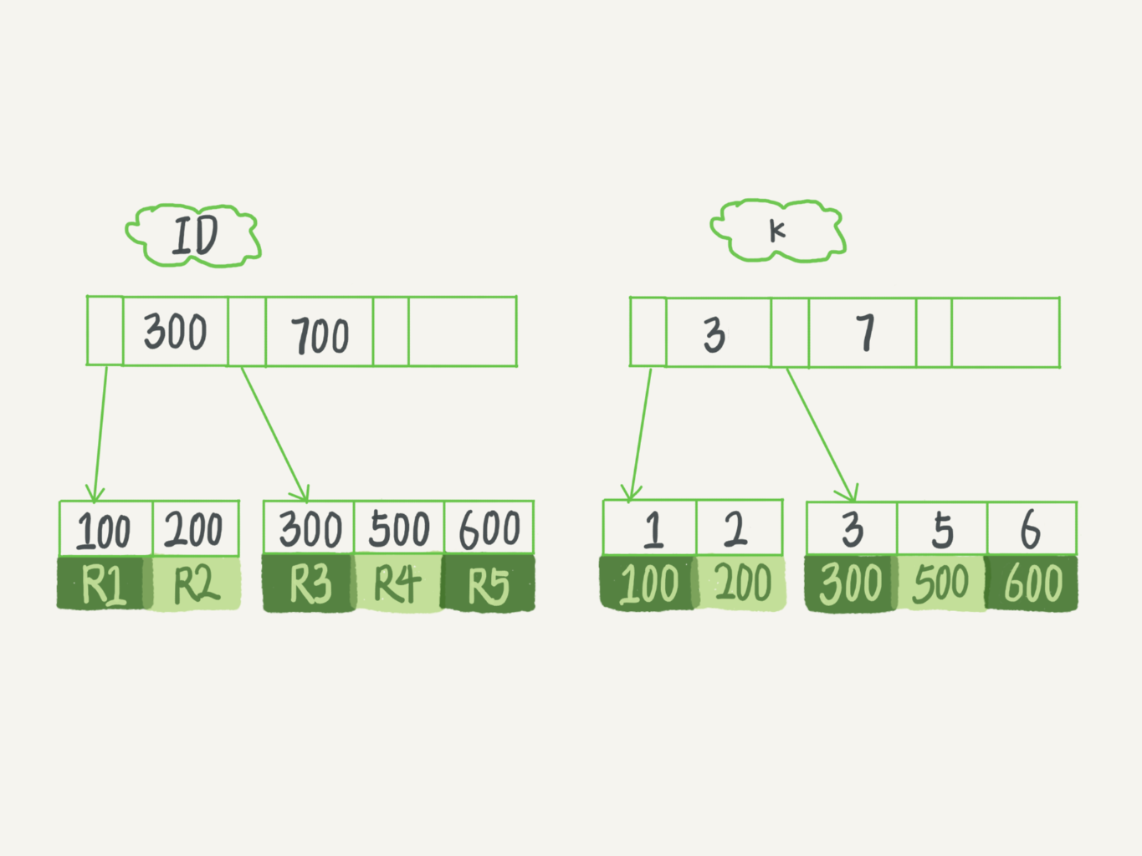
k int NOT NULL DEFAULT 0,

s varchar(16) NOT NULL DEFAULT '',

index k(k))

engine=InnoDB;

insert into T values(100,1, 'aa'),(200,2,'bb'),(300,3,'cc'),(500,5,'ee'),(600,6,'ff'),(700,7,'gg');



语句的执行流程

1. 在k索引树上找到k=3的记录，取得ID=300;
2. 再到ID索引树查到ID=300对应的R3;
3. 在k索引树去下一个值5，取得500；
4. 再回到ID索引查到ID=500对应的R4;
5. 在k索引值取下一个K=6,不满足条件，退出循环

在这个过程中，回到主键索引树搜索的过程，我们称为回表，这个查询过程读了K索引树的3条记录（步骤1,3和5），回表两次（步骤2和4）

由于查询结果所需要的的数据只在主键索引上，所以不得不回表，那么，有没有可能经过索引优化，避免回表呢？

**覆盖索引**

**由于覆盖索引可以减少搜索的次数,显著提升查询性能,所以使用覆盖索引是一个常用的性能优化手段**

需要注意的是:在引擎内部使用覆盖索引在索引k上其实读了三个记录，R3-R5(对应的k上的记录项)，但是对MySQL的Server来说，他就是找引擎拿到了两条记录，因此MySQL认为扫描行数是2；

基于覆盖索引的说明，我们来讨论一个问题，在一个市民信息表上，**是否有必要将身份证号和名字建立联合索引？**

假设这个市民表的定义是这样的：

CREATE TABLE `tuser` (

`id` int(11) NOT NULL,

`id\_card` varchar(32) DEFAULT NULL,

`name` varchar(32) DEFAULT NULL,

`age` int(11) DEFAULT NULL,

`ismale` tinyint(1) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `id\_card` (`id\_card`),

KEY `name\_age` (`name`,`age`)

) ENGINE=InnoDB

身份证号是唯一标识，如果根据身份证号查询市民信息的需求，只要在身份证号字段上建立索引就够了，在建立一个（身份证号，名字）的联合索引，是不是浪费空间。

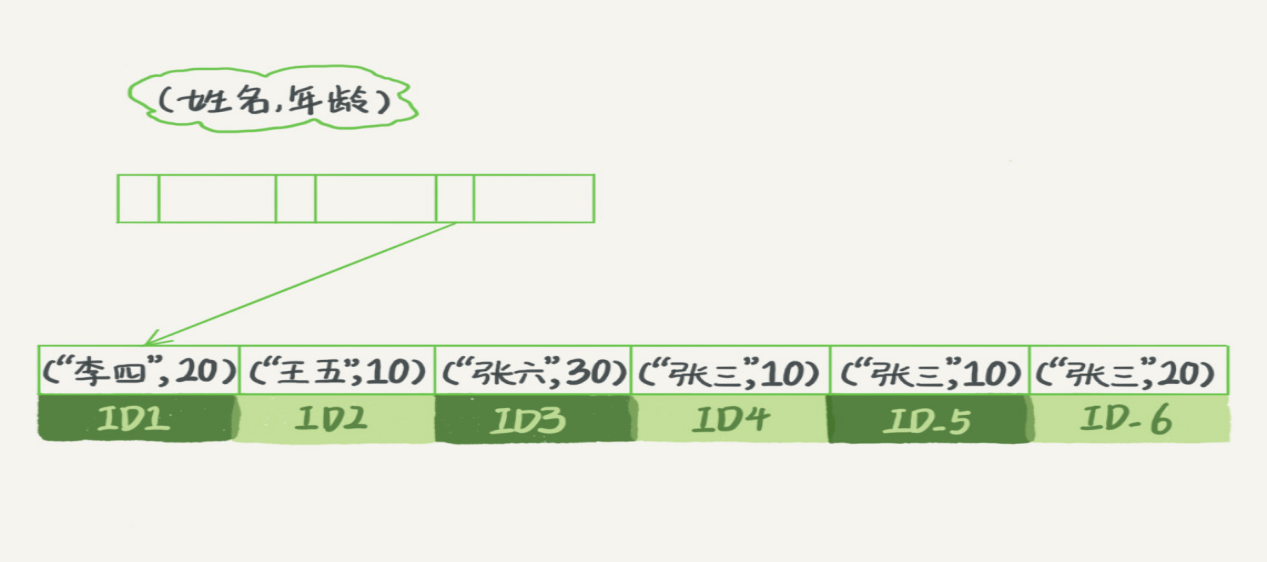
如果有个高频的请求，要根据市民的身份证号查询他的名字，这个联合索引就有意义了，他可以在这个高频请求上用到覆盖索引，不再需要回表查整行记录，减少语句的执行时间

索引字段的维护是需要代价的，在建立冗余索引来支持覆盖索引需要权衡考虑

**最左前缀原则**

看到这里有个疑问，如果为每一个查询都设计一个索引，索引太多了。如果我们发现按照市民的身份证号，去查他的家庭地址呢？虽然这个查询需求在业务中出现的概率不高，但总比不能让他走全表扫描？反过来说， 单独为一个不频繁的请求创建一个（身份证，地址）的索引有感觉有点浪费，应该怎么做呢？

为了直观的说明这个概念，我们用（name,age）这个联合索引分析



可以看到，索引是按照索引定义里面出现的字段顺序排序的

当你的逻辑需求是查到所有的名字是“张三”的人时，可以快速定位到ID4,然后向后遍历得到所需要的结果。

如果要查的所有名字第一个字是“张”的人，SQL语句的条件是“where name like ‘张%’”，此时也能够用上这个索引，查找到第一个符合条件的记录是ID3,然后遍历，知道不满足条件为止。

不只是索引的全部定义，主要满足最左前缀，就可以利用索引来加速检索，这个最左前缀可以是联合索引的前N个字段，也可以是字符串索引的最左M个字符。

基于上面的说明，**在建立联合索引时，如何安排索引内的字段顺序**

评估的标准是---索引的复用能力，因为可以支持最左前缀，所以当已经有了（a,b）这个联合索引，一般就不需要单独在a上建立索引了，因此，**第一原则是，如果通过调整顺序，可以少维护一个索引，那么这个顺序往往就是需要优化考虑采用的。**

所以现在知道了，这段开头的问题，我们要为高频请求创建（身份证号，姓名）这个联合索引，并用这个索引支持“根据身份证号查询地址”的需求。

那么，如果既有联合查询，又有基于a,b各自的查询呢？查询条件只有b的语句，是无法使用（a,b）这个联合索引的，此时不得不维护另外一个索引，也就是同时维护（a,b）(b)这两个索引。

此时要考虑的**原则是空间**了，不如上面这个市民的情况，name比age字段大的，那就建议创建一个（name,age）和（age）单字段索引。

**索引下推**

还是以市民表的联合索引（name,age）为例，如果有一个需求，检索出表中“名字第一个字是张，而且年龄是10岁的所有男孩”

select \* from tuser where name like '张 %' and age=10 and ismale=1;

知道了最足前缀索引原则，这个语句搜索索引树的时候，只能用“张”，找到第一个满足条件的记录ID3,然后判断其他条件是否满足，

在MySQL5.6之前，只能从ID3开始一个个回表，到主键索引上找出数据行，再对比字段

而MySQL5.6引入索引下推优化，可以在索引遍历过程中，对索引包含的字段先判断，直接过滤不满足条件的记录，减少回表次数。

是这两个过程的执行流程图。

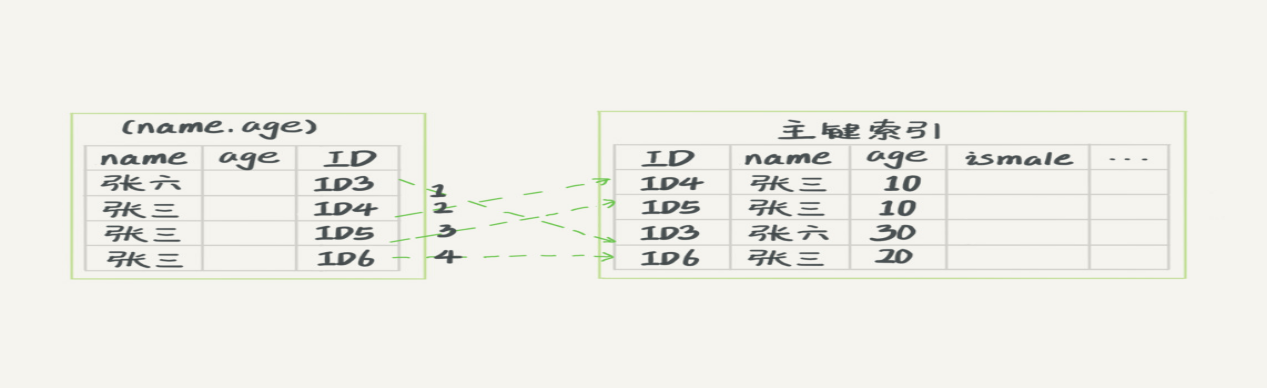


图3 无索引下推执行流程

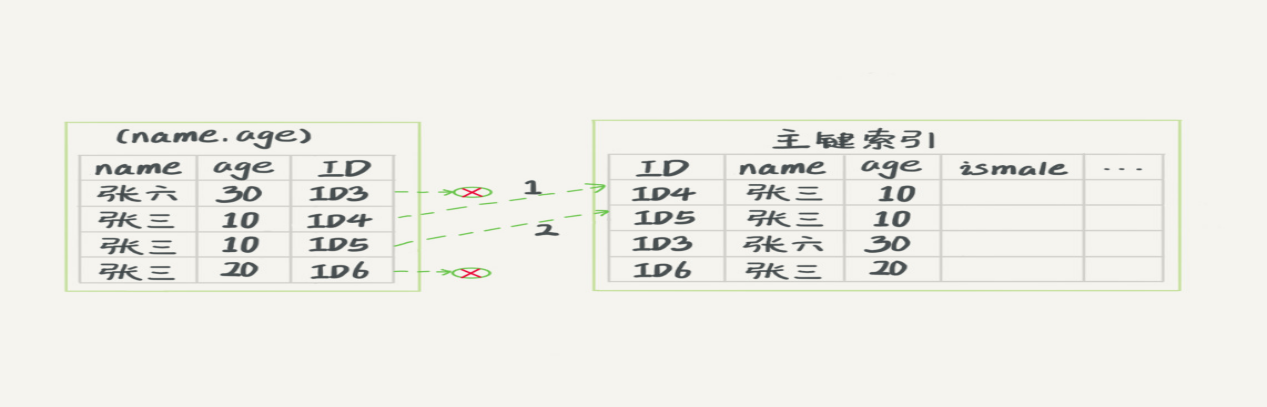


图 4 索引下推执行流程

在图3和图4两个里面，每个虚线箭头表示回表一次

图3中，在（name，age）索引里面特意去掉了age的值，这个过程InnoDB并不会去看age的值，只是按顺序把“name”第一个字‘张’的记录一条条取出来回表，因此，需要回表4次。

图4和图3的区别是，InnoDB在（name,age）索引内部就判断了age不等于10,对于不等于10的记录，直接判断并跳过，只需要对ID4,ID5这两条记录回表取数据判断，就只需要回表2次。

**讨论题**

DBA小吕在入职新公司时，发现自己接手维护库里面有这么一个表，表结构定义这样

CREATE TABLE `geek` (

`a` int(11) NOT NULL,

`b` int(11) NOT NULL,

`c` int(11) NOT NULL,

`d` int(11) NOT NULL,

PRIMARY KEY (`a`,`b`),

KEY `c` (`c`),

KEY `ca` (`c`,`a`),

KEY `cb` (`c`,`b`)

) ENGINE=InnoDB;

公司同事告诉他，由于历史原因，这个表需要a,b做联合索引，这个小吕理解

但是，学过本章的内容的小吕纳闷了，既然主键包含了a,b连个字段，那意味着单独在字段c上创建一个索引，就已经包含了3个字段啊，为什么要创建“ca”,“cb”这个连个索引？

同时告诉他，是因为他们的业务里面有这样的两种语句

select \* from geek where c=N order by a limit 1;

select \* from geek where c=N order by b limit 1;

问题是：这位同事的解释对吗，为了这个两个查询模式，这个两个索引是否都必须的？为什么呢？

Select ... from geek where c= N order by a走，ca,cb都能定位到满足c=N主键

而且主键的聚蔟索引本身是就是按order by a，b排序的，无序重新排序，所以ca可以去掉

Select... from geek where c=N order by b

这条sql如果只有c单个字段的索引，定位记录可以走索引，但是order by b的顺序与主键顺序不一致，需要额外排序， cb索引可把排序优化

InnoDB会把主键字段放到定义字段后面

所以，当主键是（a,b）的时候

定义为c的索引，实际上是（c,a,b）

定义为（c,a）的索引，实际上是（c,a,b）

Ps 定义为（c,b）的索引，实际是（c,b,a）