参考：

MySQL实战45讲

前言

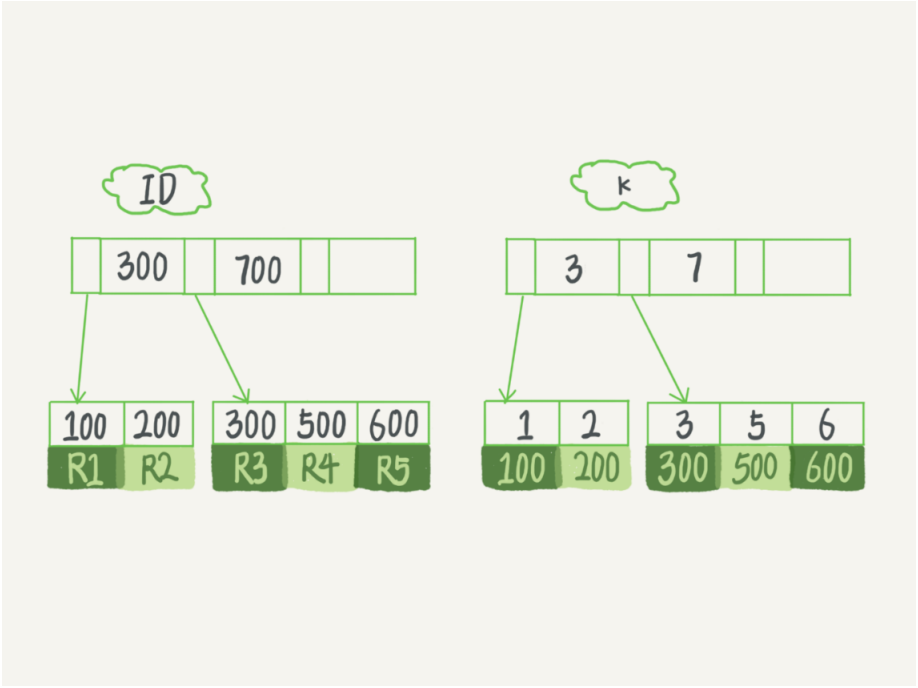
索引都知道能提高查询效率，稍微知道的多点呢，知道索引就像一本字典的目录，也就是为什么能提高查询效率的一个抽象的认知，本篇就围绕索引的原理展开去整理。

InnoDB的索引模型

在InnoDB中，表都是根据主键顺序以索引的形式存放的，这种存储方式的表称为索引组织表。所以本质上，每张表都至少有一个索引——主键索引，有人要问我没有主键咋办，放心，看似没有主键的表，InnoDB都会自动生成一个主键作为标识，只是我们看不到罢了。

回归正题，InnoDB使用的是B+树索引模型，换句话说，每一个索引在InnoDB中都对应一个B+树。

我们假设有张表中R1~R5的(ID,k)对应(100,1)(200,2)(300,3)(500,5)(600,6)，ID是主键，k也建有索引。针对这两棵索引树的示意图如下。



从图中我们可以清晰看出，主键索引是直接绑定了记录值，非主键索引绑定的是主键值，那么我们不难推测。

```
select * from T where k=5;
```

这一条SQL语句的明显用到了索引，肯定是先检索k索引树，然后得到ID值再检索主键索引树得到对应的记录值。（InnoDB中，主键索引又称聚簇索引，非主键索引又称二级索引）

因此，使用非主键索引时需要多扫描一棵索引树，所以我们应在可能场景下都避免使用二级索引。

覆盖索引

还以上述的图为例。

```
select * from T where k=3;
```

这条SQL的执行流程是啥样的呢？

1.在k索引树z找到k=3的记录，取得ID=300；

2.再到ID索引树找到ID=300的记录R3；

3.在k索引树娶下一个值k=5，不满足条件，结束（涉及到B+树按序排序的问题）。

因为索引对应的是二级索引，这个过程中经历了一次回表（回到主键索引树搜索的过程），那么显然效率就低了嘛。那么我们能不能优化下呢，假设，我现在只想取得ID值，那么我们可以这样写SQL。

```
select ID from T where k=3;
```

这时候你会发现，我们的k索引树存放的记录不就是ID嘛，InnoDB也不是蠢蛋，在k索引树找到k=3的记录取到ID=300就是取到值了，就不会再去回表了，像这样索引已经覆盖了我们的查询需求，就是覆盖索引。

当然，覆盖索引还有个方式，联合索引的中的A和B，查询条件为A，结果集是B这样的，也属于覆盖索引。

设计一个案例。

比方市民表这样定义。

```
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
```

我们有个频繁操作的需求，根据身份证号码查询姓名，你想想如果我们给身份证号码和姓名建立一个联合索引，你会发现这个查询效率提高了好多。当然并没有这个频繁操作需求或者这个需求操作次数很少，建这个联合索引就有点冗余了。

最左前缀原则

既然说到联合索引的问题，我们就来展开说说，经过上面的叙述，索引的好处那是显而易见的了，那么是不是我们就得每个查询都去建个索引呢，那显然不行啊，你想想，你一本字典1000页翻完了目录还没结束那场面。那么InnoDB也是智能的，针对联合索引，InnoDB支持很好的复用，往往你一个联合索引设计的好，能被当成好多个索引来使用，适用的查询场景就多了。

其实就是允许联合索引中的一个到多个组合者也能使用，或者说联合索引的部分子集你也可以当作一个索引使用，为什么强调部分呢，这不是做数学题求子集，InnoDB也不能让你胡来啊，到时候它识别反而更耗费时间，因此有个规则，就是最左前缀原则。

比方(A,B,C)，你的查询条件是A，AB，ABC都可以使用这个索引去加速查询。

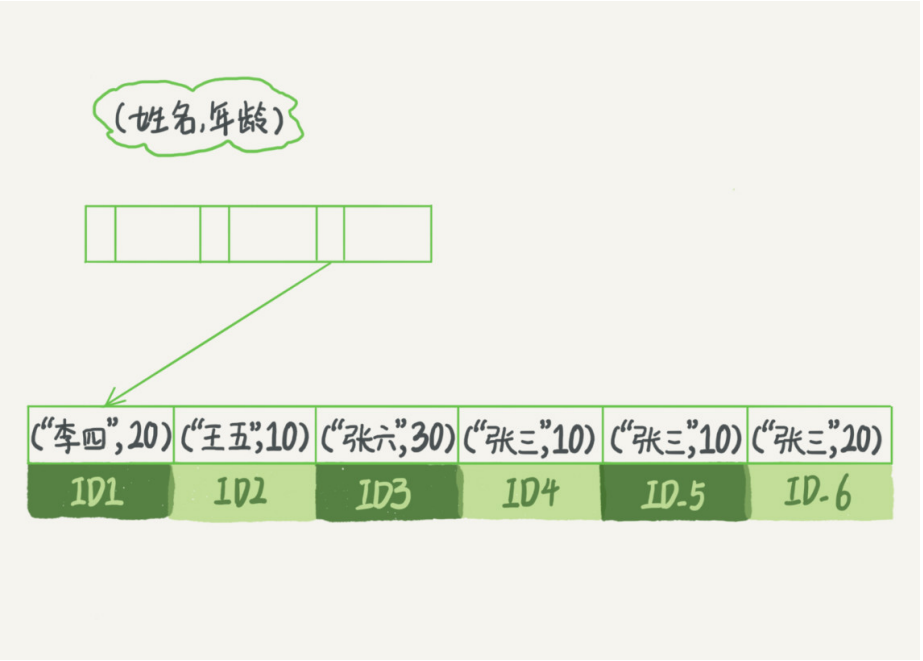
所以往往建立联合索引时，列的顺序安排的好可能都能少维护一个索引，比方我得条件是BA，我们把上述索引改为(B,A,C)，就能使用了，总的来说还是看具体的案例。

那么还有种情况，索引(A,B)，但我又同时有基于A和B自身的查询，这时候肯定还得单独维护个A或B的索引，看似那么这两个的联合索引顺序随意就好了呗，不然，还是有可以优化的地方的。

比如A的字段比B的大，这时候我们维护(A,B)，(B)就比另一种方式好，为啥呢，索引也是占空间的啊，A大自然不如用B构建索引的空间小啊，索引在顺序无法判断的时候，空间也是一个不错的考虑思路。

索引下推

以上述的市民表为例，有个联合索引(name,age)，有个需求“名字第一个字是张，年龄10岁的所有男孩”。



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

那我们肯定会搜索(name,age)索引树先找到第一个满足条件的ID3，然后就会回表，然后继续。

MySQL5.6之前确实就是这么做的，因为模糊查询，并不能用覆盖索引的那一套。

但是MySQL5.6之后就引入了索引下推的概念。

其实按照我们人为的想法，既然都模糊查询到了name，并且也有明确的age条件，而且联合索引中也有age，为啥不先筛选掉呢，索引下推就干了这个事情，我们在索引内部就判断了age的值是否符合，符合才继续回表，否则就直接pass掉了，这样的优化明显的更符合时代的发展。