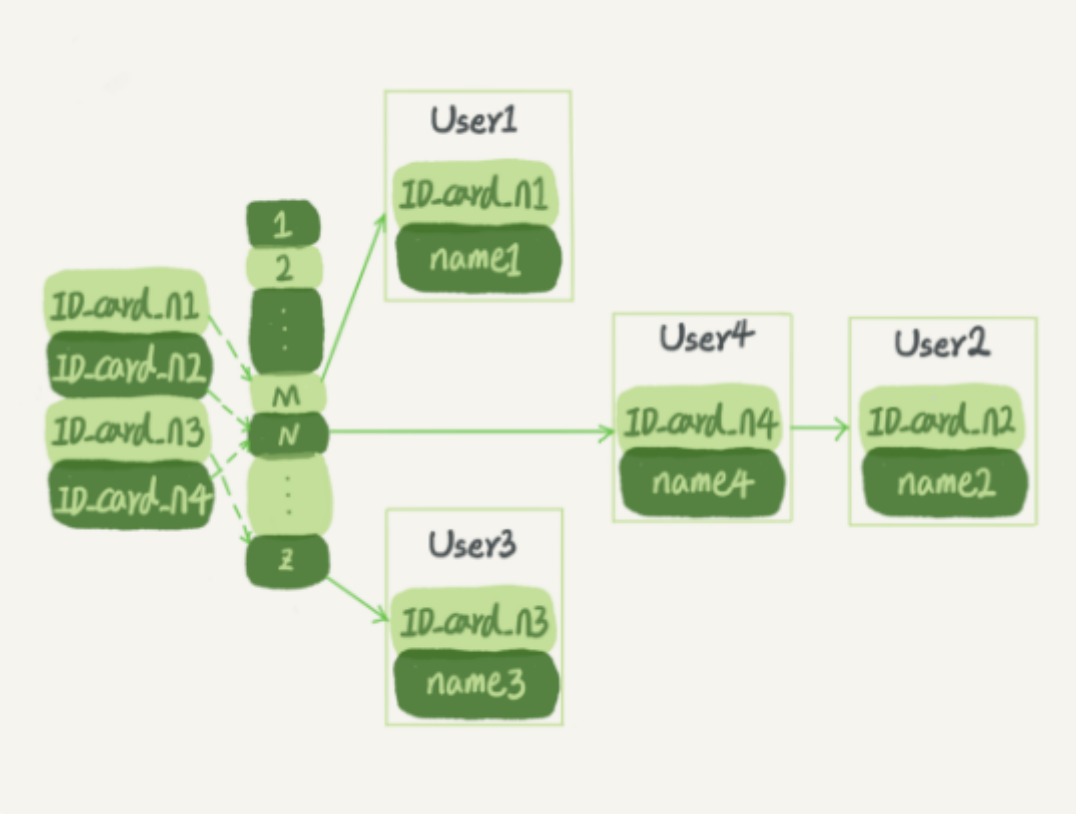
1.索引的常见模型

索引的出现是为了提高查询效率，但是实现索引的方式却有很多种。目前常见的有三种方式：

- 哈希表
- 有序数组
- 搜索树

1.1 哈希表

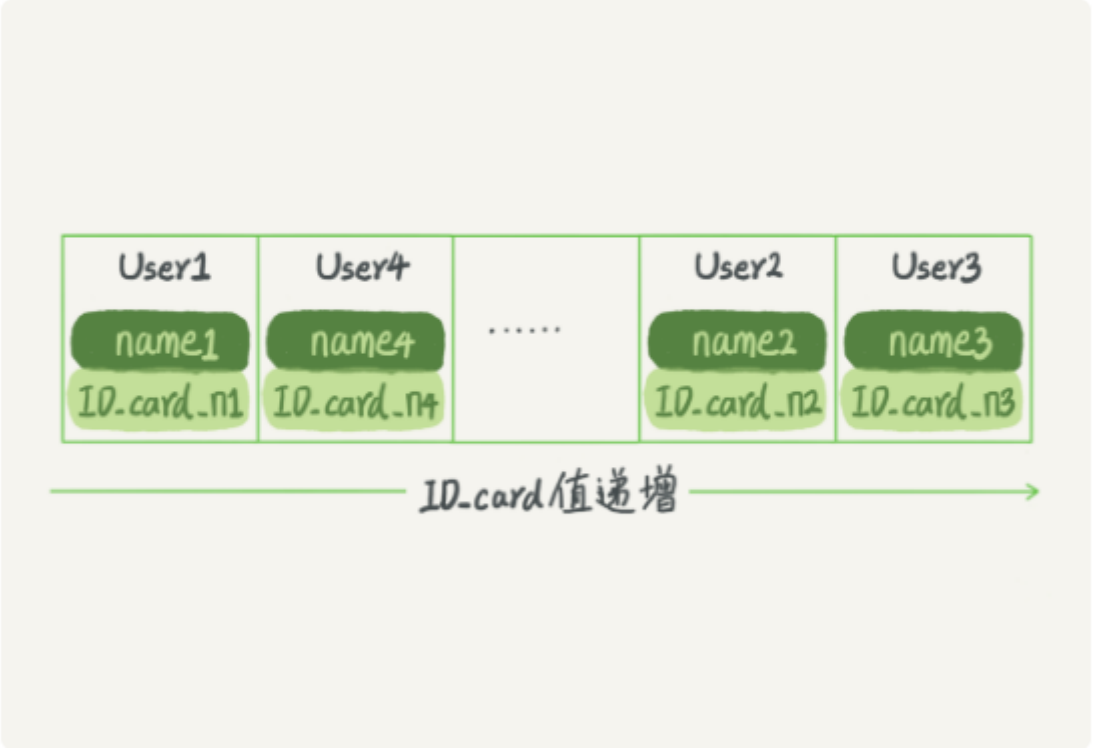
哈希表是以key-value来存储的数据结构，通过哈希函数把key换算成一个位置，再把value存储在数组对应的位置上。当数据变多之后，肯定会产生哈希碰撞，得到的是同一个值，这个时候在对应位置拉出一个链表解决。



如图所示，这是一个id和name构成的哈希表，如果知道具体的id值或name值查询效率会非常快。因为存储的数据不是有序的，所以对于范围查询来说，就必须遍历整个表，效率很低。

1.2 有序数组

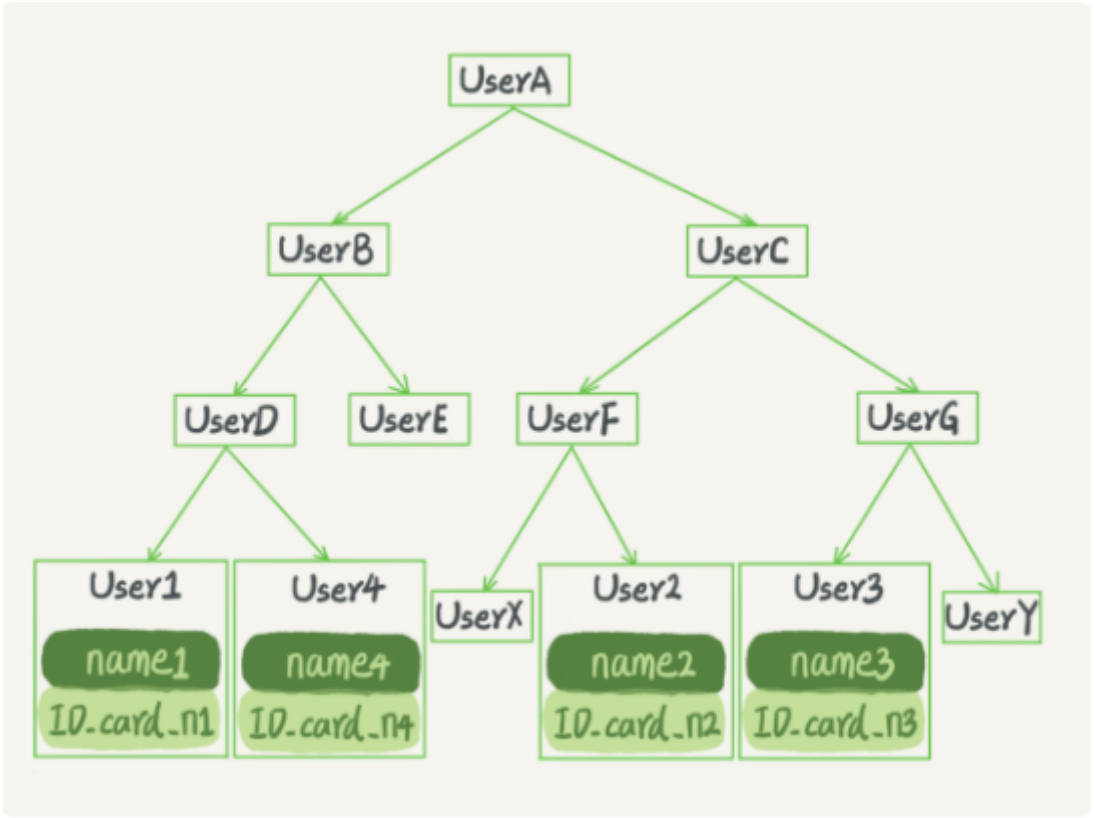
有序数组就很好的解决了哈希表范围查询效率低的问题，整个数组是有序的，对于等值查询和范围查询效率都很高。



如图所示，在id值按照大小顺序排列时，等值查询和范围查询用二分法都可以很快查出结果。但是有序数组如果需要插入数据时，需要移动后面的数据以保证数组的有序性，开销很大。所以有序数组适用于静态表。

1.3 搜索树

二叉搜索树在查询和插入的效率中找到了一个平衡。



如图所示，左儿子小于父节点，父节点又小于右儿子。查询的时间复杂度时O(log(N)),插入数据的时间复杂度也是O(log(N)).但二叉树还不能满足实际需求，假如一个100万节点的二叉树，树高20，随机读取磁盘需要10ms，这时查询一个数据需要10x20ms，这个效率还是不足，所以实际InnoDB是多叉树，大约是1200，树高4层时就足以存储17亿数据，如果查询一个值，最多只需要访问3次磁盘。

2.InnoDB的索引模型

在InnoDB引擎中，表都是根据主键顺序的形式存放的，这也叫做索引组织表，所有的数据都是存储在在叶子节点的，叶子节点上存的是page(页)。 InnoDB每一个索引对应一颗B+树。主键索引的叶子节点存的是整行数据，也叫做举簇索引。非主键索引叶子节点上存储的是主键索引值，也叫做二级索引，辅助索引，非举簇索引等。由于非主键索引上存的是主键索引的值，所以用非主键索引查询数据库时会去扫描主键索引的B+树，这个过程叫做回表，在开发中应尽量使用主键索引查询数据。

3.索引维护

B+树为了维护索引的有序性，在插入数据时会调整树结构。如果数据存储的数据页满了，还会开辟新的数据页，移动部分数据过去，造成性能损耗，还有可能使本应放在一个数据页的数据，分裂到两个页存储，造成资源的浪费。基于这个特性，我们一般建议数据库尽量用自增主键，每次插入新数据都是追加，不涉及到索引树结构的改变。主键索引的选择也很重要，假如用20字节的字符串做主键索引，每个二级索引的叶子节点存的都是主键索引，20字节，相比int类型的主键只占4字节，显然int类型的主键索引会节省空间。当然这个也跟业务场景有关，如果存在这样的业务数据，是K-V的形式，只有一个索引，那么也可以用这个K做主键索引，查询效率是最高的。

4.覆盖索引

二级索引上存的是主键索引的值，如果需要查询数据，就需要先用二级索引查出主键索引再去主键索引树查到数据，但是假如查询的数据就是主键id的时候，因为已经存储在二级索引的叶子节点，所以不需要回表就可以查到，这个就叫覆盖索引。覆盖索引也是常用的性能优化手段之一。

5.最左前缀原则

索引在检索的过程中，如果满足联合索引的最左N个字段或者是字符串索引的最左M个字符的时候，都是可以使用到索引的。设计联合索引的时候需要充分考虑最左前缀原则，假如存在（a,b)索引的情况下，基本上可以不用创建a的单独索引，所以联合索引的顺序就显得至关重要。假如存在需要对b单独检索的情况，那么就需要建一个b的独立索引，但具体问题需要具体分析。如果b占用的空间比a大，那么这里显然是建立（b，a）联合索引，再单独建立一个a索引占用的资源是最少的。

6.索引下堆

在mysql5.6之前对于搜索条件中索引存在的字段并不会去判断，而是会回表的时候再进行判断，但是mysql5.6之后进行了索引下堆优化。假设索引中存在了足够检索条件判断的数据，那么这个时候就不会回表判断，而是在索引遍历过程中进行判断是否满足这个检索条件。