1 索引

1.1什么是索引

Mysql 官方对于索引的定义是：索引可以帮助 mysql 高效获取数据的数据结构。即索引是数据结构。数据库在执行查询的时候，如何没有索引存在的情况下，会采用全表扫描的方式进行查找。如果存在索引，则会先去索引列表中定位到特定的行或者直接定位到数据，从而可以极大地减少查询的行数，增加查询速度。

可以类比为一部字典开头的目录。

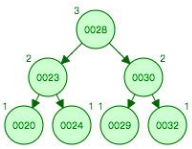
1.2索引数据结构

二叉树
红黑树
Hash 表
B 树
常见的数据结构，究竟哪种数据结构更适合在数据库的索引中使用呢?
https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

1.2.1二叉树（平衡）、红黑树

未加索引，则依次扫描查找

id	name	age
1	zhangsan	32
2	lisi	24
3	wangwu	28
4	zhaoliu	29
5	xiaoming	20
6	hanmeime	23
7	lilei	30



未添加索引之前，如果需要查找某行数据，则需要根据查询条件遍历全表去查找。添加索引之后，查找某行数据，则只需要几次查询即可查到该索引数据，同时二叉树中的每一个元素也保存了相应行数据的磁盘地址，通过该磁盘地址，便可以定位到对应行的数据。但是如果

二叉树作为索引的话会存在什么样的问题？

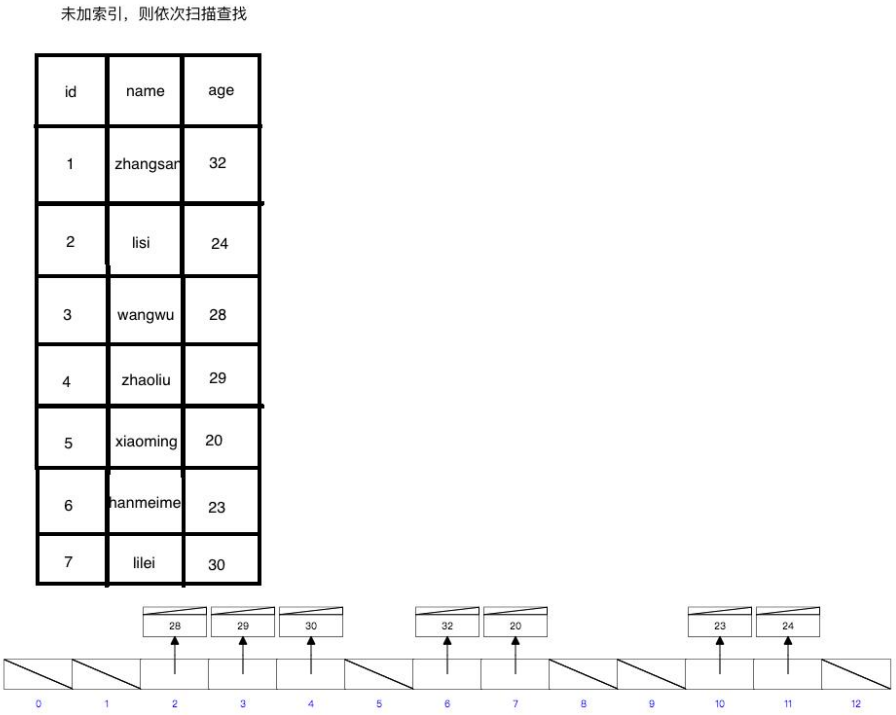
1.2.2 Hash 表

也叫散列表，根据相关的 key 而直接访问的数据结构。做法很简单，把 key 通过一个固定的运算转换成一个数字，然后将这个数字对数组的长度取余，最终的结果就当做数组的下标。对应的数据就放在该下标处。

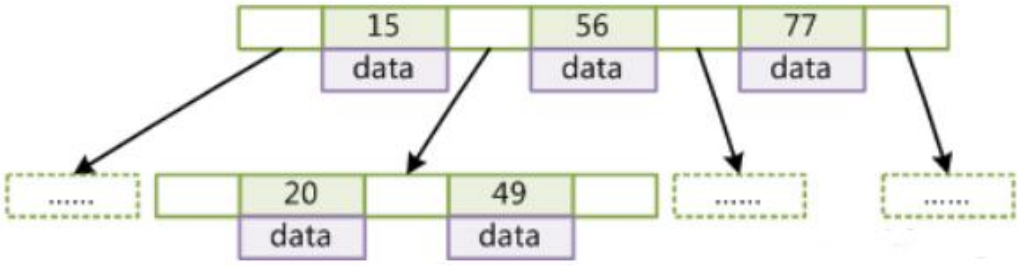
如果 hash 表作为索引，其查询效率也是很高的。但是 hash 会普遍用于数据库的索引上面来吗？

Hash 索引会有以下几种问题：

1. hash 索引仅能够查找=，in 等查找，无法进行范围查找
2. hash 索引无法用来进行排序（经过运算过后的数字大小和原本数字大小没有关系）
3. 如果设置了若干字段的一个组合索引，那么 hash 索引无法利用部分字段进行索引查找，比如设置了用户名、密码、邮箱的联合索引，那么无法使用用户名、密码来通过索引查找。
4. Hash 索引很有可能会导致不同的数据经过运算之后得到相同的 hash 值，因此即便找到了对应的 hash 值所在的下标，仍有可能需要进行再次扫描表数据。
5. 如果存在大量的 hash 值相等的情况，那么 hash 索引此时的查询性能不一定优秀



1.2.3 B 树



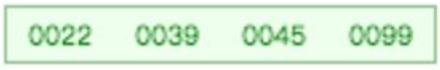
B 树也叫 B-树。是一种多路平衡查找树。B 树的定义如下：
对于一颗 m 阶的 B 树而言（阶数表示一个节点最多有多少个孩子节点）：

- 每个节点最多有 m-1 个 key
- 根节点最少有一个 key，两个子女
- 非根节点包含的 key 个数满足： ⌈m/2⌉ - 1 <= j <= m - 1
- 每个节点中的 key 都是按照从小到大的顺序排列
- 所有叶子节点位于同一层

1.2.3.1 B 树插入

对于一颗 5 阶的 B 树，有以下几个特点：
每个节点至多有 5 个孩子子节点
每个节点内至多有 4 个 key，至少 2 个 key
每个节点内有 n 个 key，以及 n + 1 个指针

在 B 树种插入 39，22，99，45

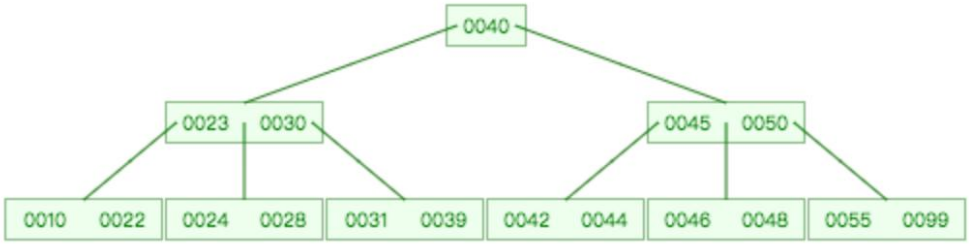


继续插入 55，节点内 key 个数为 5 个，要发生裂变

以中间值进行分开，如图所示
继续插入数据 10，23，40，同样左下的节点超过 4 个 key 会发生裂变

继续插入 24，28，30，42，44，50

继续插入 46，48，最右侧会超过 4 个 key，然后发生裂变，中间值向上，进入父节点，但是这个时候，父节点也会超过 4 个 key，所以进一步以中间值向上裂变，形成下图所示



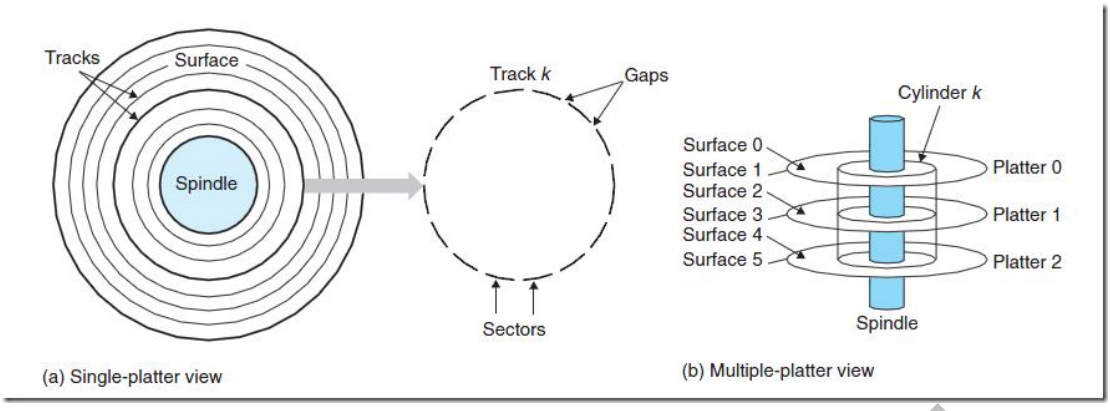
从图示可以看出，如果想查询某个数字，则需要经过三次查询即可查到对应的数据。索引是存在于磁盘中的，也就是说要经过三次磁盘 IO 操作便可定位到对应的索引值。

问题一：B 树相较于平衡二叉树或者红黑树，最大的优势在什么地方？？

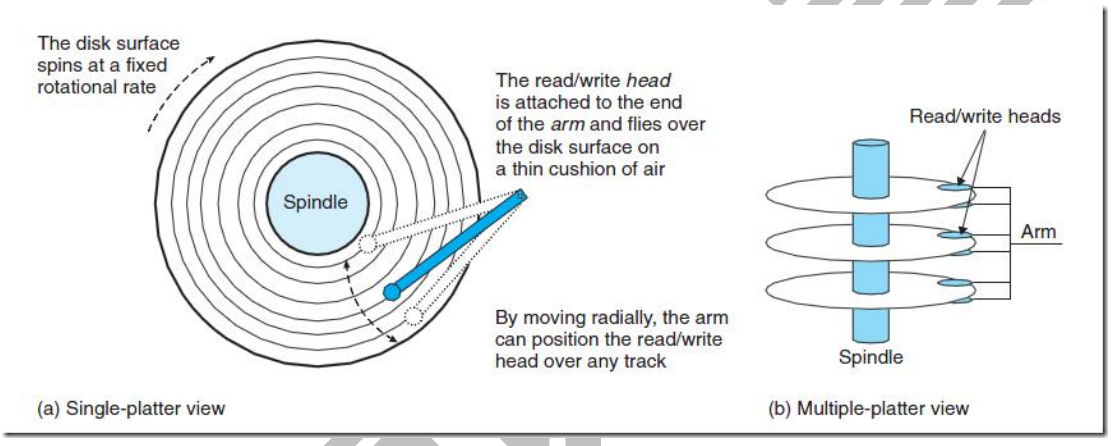
尤其是数据量越多，体现越明显。
对于 B 树而言，如果节点内存储的索引数量越多，那么即便 B 树的高度只有三层或者四层，也可以存储千万条以上的数据。
一般情况下,B 树的高度如果是 3，那么就需要进行三次查询，也就是说需要经过三次磁盘 IO，查询的限速步骤主要在于磁盘 IO，那么

问题二：如果将千万条数据全部存放在一个节点内，不是只需要一次磁盘 IO 就可以找到对应的数据了吗，为什么不采用这样的方式呢?

首先要清楚磁盘和内存是如何交互的。磁盘的结构如图:



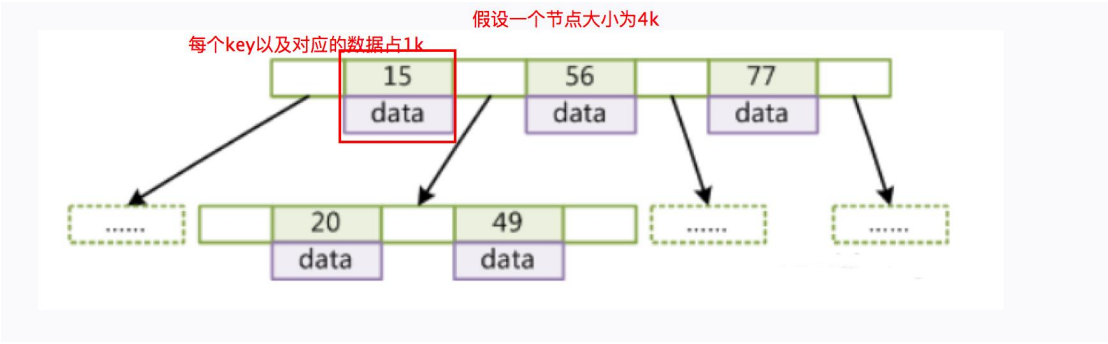
磁盘由盘片组成。盘片有两面，称之为盘面 surface。盘面上覆盖磁性材料。中间是一个可以旋转的轴 Spindle，使得整个盘面能够旋转。通过速率为 5400 转每分钟或者 7200 转每分钟。每个盘面又由一系列的同心圆组成，称之为磁道 track。磁道又被划分为一组组扇区 sector。扇区的大小大概都相等，为 512 字节。



当从磁盘中读取数据时，利用读写头找到对应的磁道，这个步骤称之为寻道。找到磁道后，盘片开始转动，磁道上的每个位都可以被磁头感知到，然后读取到其内容，对磁盘的访问整个过程可以分为寻道时间和访问时间。一般情况下，寻道时间较慢。

由于磁盘存取的速度比内存慢很多，所以磁盘读取时，通常情况下并不是按需读取，而是会预读一部分数据。预读的长度通常情况下为一个页的整数倍。一个页一般情况下大小为 4k。也就是说一次磁盘 IO 通常只会读取 4k 的几倍。因此，把全部数据写入到一个节点中，也并没有太大用处，因为一般只会读取 4k 或者 4k 的几倍。

数据库的实现者也利用磁盘预读的这一点，将一个节点的大小设为一个页的大小或者一个页大小的几倍。



假设一个节点的大小是 4k，然后每个 key 以及对应的数据加起来一共 1k，那么该节点可以存放多少个 key，即索引字段？
那么，如何才能让每个节点存储更多的索引字段呢？数据库底层采用的是 B 树吗？

1.2.4 B+树

数据库底层采用的其实是 B+树来作为索引。它可以看成是 B 树的变种。具有以下特点：

- 非叶子节点不存储 data，只存储 key
- 所有的叶子节点存储完整的一份 key 信息以及 key 对应的 data
- 每一个父节点都出现在子节点中，是子节点的最大或者最小的元素
- 每个叶子节点都有一个指针，指向下一个数据，形成一个链表

特点：B+树由于非叶子节点不存储数据，仅在叶子节点才存储数据，所以，单个非叶子节点可以存储更多的索引字段。
问题三：一个索引树最多能够存储大概多少条数据？
假设一颗索引树的高度为 3，那么一个数据库中一个页的大小是多大呢？比如在 innodb 存储引擎中，页的大小是 16kb，那么如果索引里面存放的是主键的 id 值，比如 bigint 类型，占用 8 个字节，一个指针所占用的空间大概 6 个字节。那么一个节点内大概可以存储多少个索引字段？
1170 个索引字段。如果索引的高度是 3 层，非叶子节点存储的 key 和数据加起来大概 1k 左右，那么最终总条数为 1170*1170*16=2100w 条数据。

问题四：索引为什么使用 B+树，而不使用其他数据结构比如二叉树、红黑树、hash 表、b 树等等？

1.3索引的具体实现

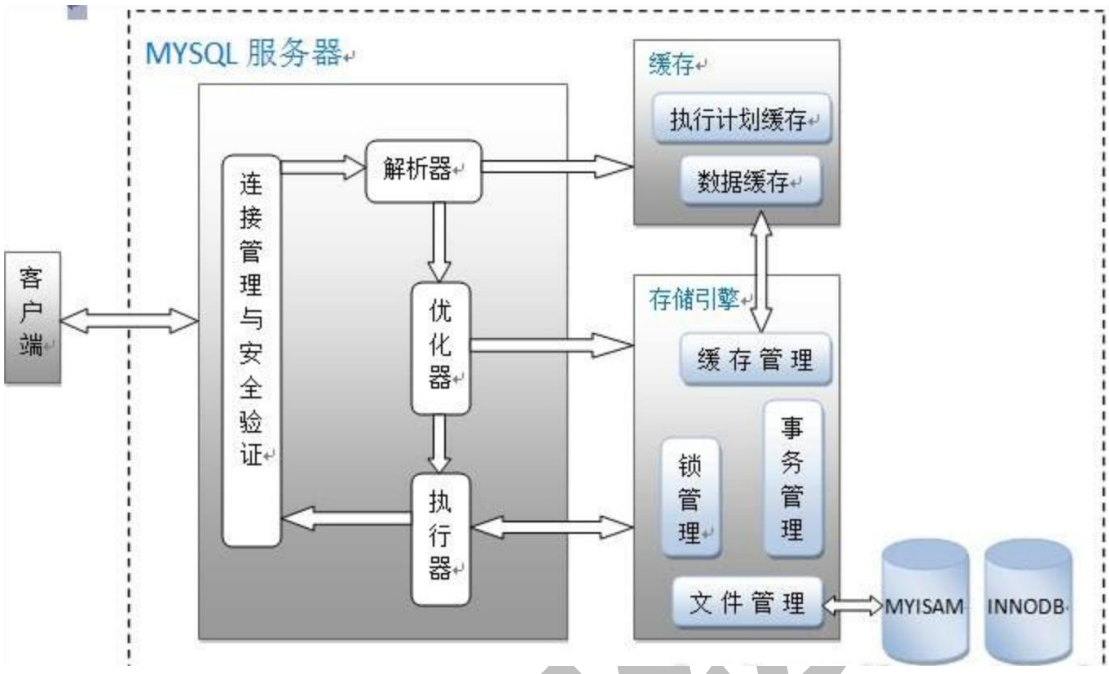
介绍索引具体实现之前，先介绍一下数据库的组成结构。为什么？
因为索引的具体实现和不同的引擎有关。
Mysql 数据库的基本组成架构为：
连接器：负责管理连接，权限的验证等。

解析器：首先 mysql 需要知道你想做什么。因此需要对输入的 sql 进行解析。首先进行词法分析，需要识别出里面的字符串代表什么意思。比如 select 代表查询，T 代表某张表，ID 代表某张表的列字段叫 id；之后进行语法分析，根据语法规则，判断输入的 sql 语句是否符合 mysql 语法。

优化器：经过解析之后，mysql 就知道你需要做什么事情了。但是在真正执行之前还需要经过优化器处理。比如当表中存在多个索引的时候，选择哪个索引来使用。或者多表关联的时候，选择各个表的连接先后顺序。

执行器：开始执行之前首先确认对该表有无执行查询的权限。如果没有，则返回错误的信息提示。如果有权限，则开始执行。首先根据该表的引擎类型，使用这个引擎提供的接口。比

如查询某表，然后利用某字段查找，如果没有添加索引，则调用引擎的接口取出第一行数据，判断结果是不是，如果不是，依次再调用引擎的下一行数据，直至取出这个表中所有的数据。如果该字段有索引，执行过程也大致相似，



所以具体的数据是保存在引擎中的。在 Mysql 中，常见的引擎有 MyISAM 和 InnoDB。

1.3.1 MyISAM 和 InnoDB 区别

1. Innodb 支持事务，MyISAM 不支持事务，对于 Innodb 中的每条 sql 语句都自动封装成事务，自动提交，影响速度
2. Innodb 支持外键，MyISAM 不支持外键
3. Innodb 是聚集索引，数据文件和索引绑在一起。MyISAM 是非聚集索引，索引和数据文件是分开的
4. Innodb 不保存表的行数，查询某张表的行数会全表扫描。MyISAM 会保存整个表的行数，执行速度很快
5. Innodb 支持表锁和行锁（默认），而 MyISAM 支持表锁。但是 Innodb 的表行锁是通过索引实现的，如果没有命中索引，则依然会使用表锁
   Innodb 表必须要有一个主键（如果用户不设置，那么引擎会自行设定一列当做主键），MyISAM 则可以没有
6. Innodb 的存储文件是 frm 和 ibd，而 MyISAM 是 frm、myd、myi 三个文件。Frm 是表定义文件，ibd 是数据文件；myd 是数据文件、myi 是索引文件

如何选择？

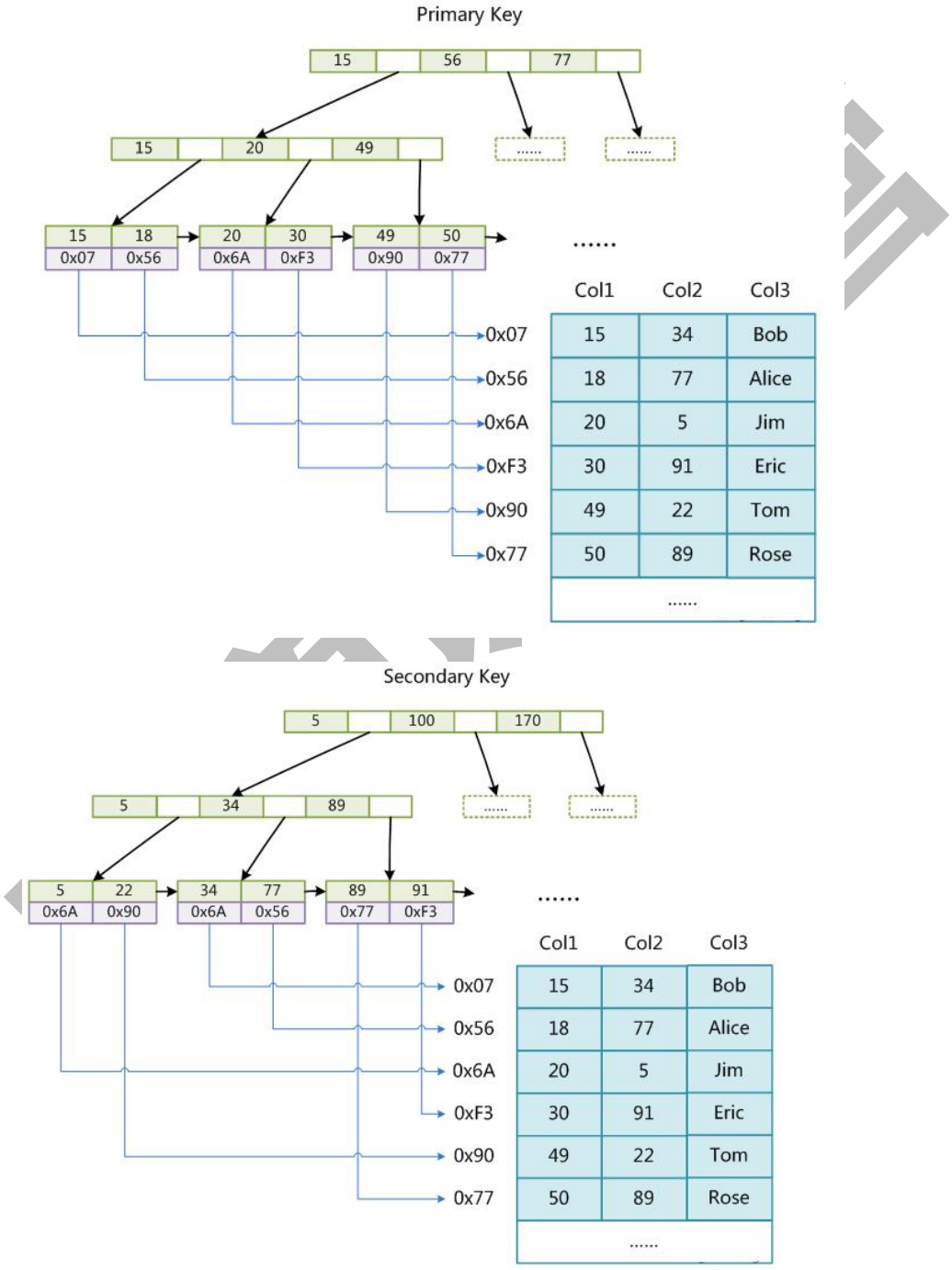
是否需要事务？如果不需要，则可以使用 MyISAM

绝大多数操作是否是查询？如果是，可以选择 MyISAM，有读也有写，则选择 Innodb

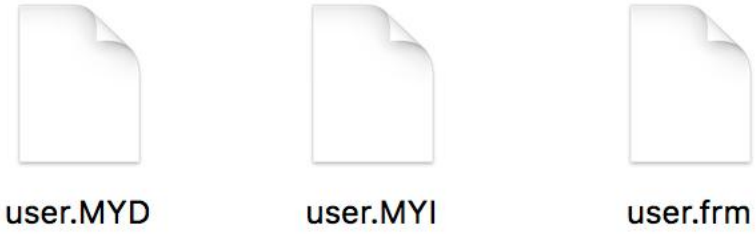
1.3.2 MyISAM 和 InnoDB 索引实现

1.3.2.1 MyISAM 索引的实现

MyISAM 的索引是非聚集索引。什么叫非聚集？MyISAM 的索引文件和数据文件是分离的。

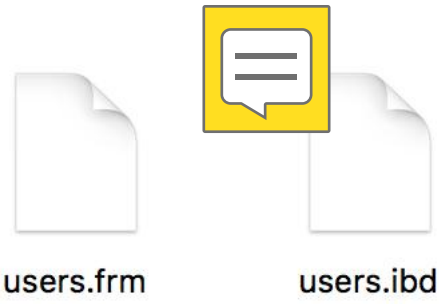


主键索引的实现方式和非主键索引（辅助索引）的实现方式并没有太大的区别。MyISAM 的文件有三个文件组成：



MYI 是索引文件。索引文件中存放的是对应数据的文件指针，接着会去 MYD 文件中去找对应指针的数据。

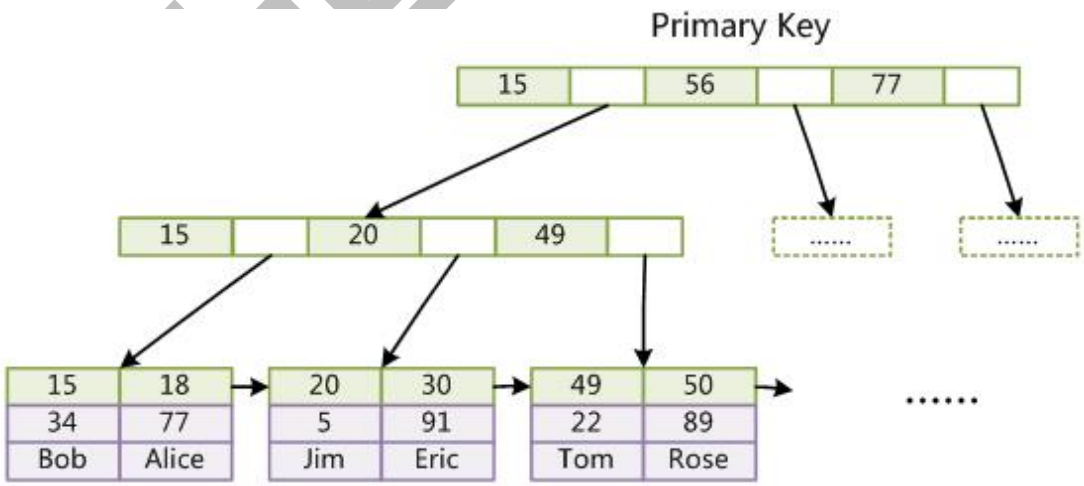
1.3.2.2 Innodb 的索引实现

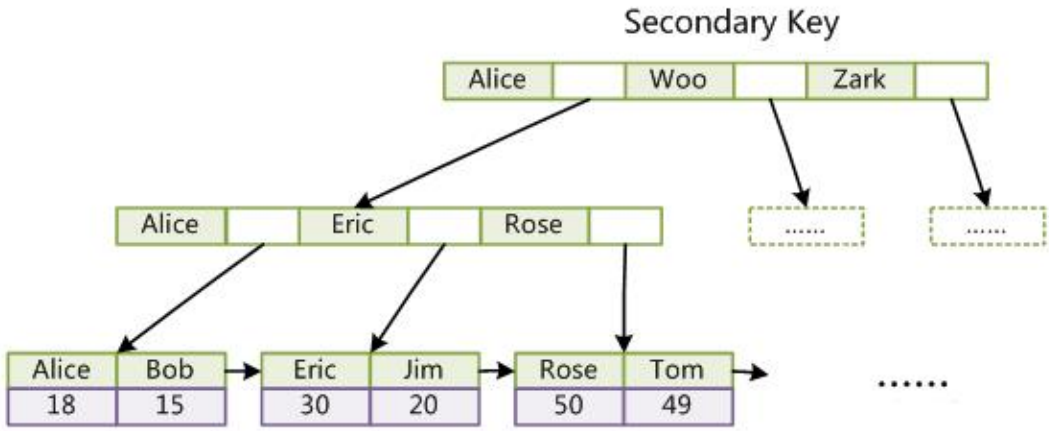


Innodb 的文件只有一个表结构文件和数据文件。数据文件本身就是索引文件。Innodb 的索引是聚集索引形式。索引和文件数据是存放在一起的。

Innodb 索引的特点：

1. 数据文件本身就是索引文件
2. 数据本身就是按照 B+树索引组织起来的一个索引文件
3. 对于主键索引的 B+树的叶子节点包含了完整的数据信息,对于非主键索引，叶子节点存放的是主键的 id





问题五：为什么 Innodb 表必须要有主键，同时推荐使用自增的整数作为主键?

问题六：为什么非主键索引叶子节点存放的是主键的值?

1.3.2.3 联合索引的实现

联合索引指的是对多列创建了索引。比如对 a，b，c 创建了一个联合索引，其实相当于创建了 a 单列索引、（a，b）联合索引以及（a，b，c）联合索引。索引最左前缀原理。

1.4索引语法

查看某张表的索引：show index from 表名;
创建普通索引：alter table 表名 add index 索引名(索引列);
创建复合索引：alter table 表名 add index 索引名(索引列 1，索引列 2);
删除某张表的索引：drop index 索引名 on 表名;

实操演示

```
create table majors(id int, username varchar(255), password varchar(255), age int);

create procedure batchInsert(in args int)
begin
declare i int default 1;
-- 开启事务(重要!不开的话,100w 数据需要论天算)
start transaction;
while i <= args do
insert into majors(id,username, password) value(i,concat("软件工程-",i),
concat("password",i));
set i = i+ 1;
end while;
```

```
commit;
end


call batchInsert(5000000);
# 复制一份 majors 表到 majori
alter table majorsi add index u_p_index(username, password);
```

无索引：

```
22
23  select * from majors where username = '软件工程-100000' and password = 'password100000';
24
25
26
27  alter table majorsi add index u_p_index(username, password);
28
29  select * from majorsi where username = '软件工程-100000' and password = 'password100000';
30
```

信息 结果 1 剖析 状态

id	username	password	age
0000	软件工程-100	password10(	(Null)

select * from majors where username = '软件工程-100000' and password = 'password100000' 只读 查询时间: 4.499s

有索引：

```
28
29  select * from majorsi where username = '软件工程-100000' and password = 'password100000';
30
```

息 结果 1 剖析 状态

d	username	password	age
000	软件工程-100	password10(	(Null)

ect * from majorsi where username = '软件工程-100000' and password = 'password100000' 只读 查询时间: 0.037s