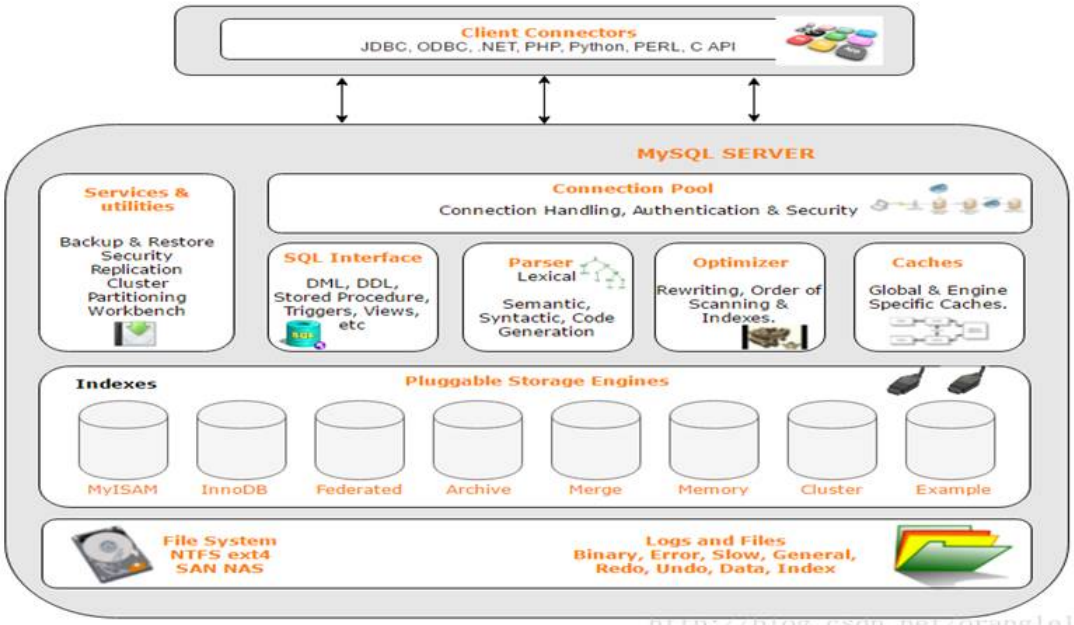
初识Mysql体系结构，理解Mysql底层B+tree索引机制

- 1.索引谁实现的
- 2.索引的定义
- 3.为什么选择B+Tree
- 4.B+Tree在两大引擎中如何体现
- 5.索引知识补充
- 6.总结及验证

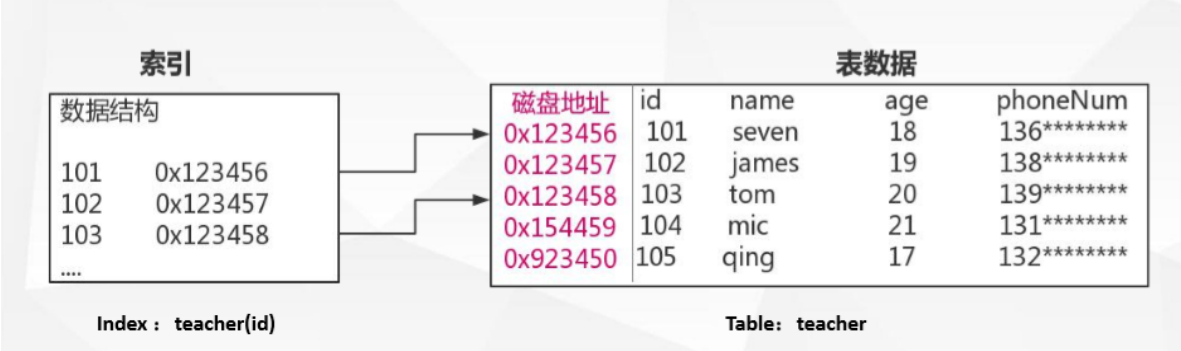
1.Mysql体系结构



> 正确的创建合适的索引 是提升数据库查询性能的基础

1.1 索引是什么?

索引是为了加速对表中数据行的检索而创建的一种分散存储的 数据结构



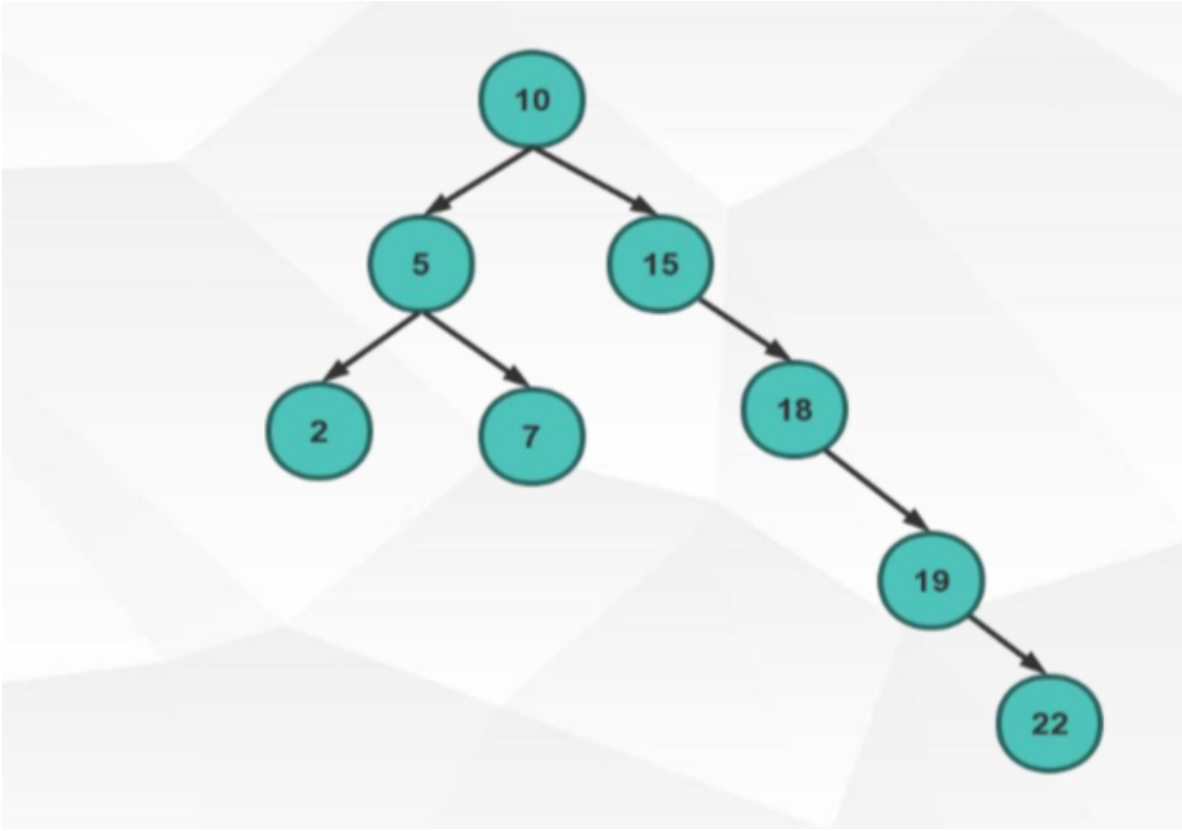
1.2 为什么要使用索引

> 索引能极大的减少存储引擎需要扫描的数据量
> 索引可以把随机IO变成顺序IO
> 索引可以帮助我们在进行分组、排序等操作时，避免使 用临时表

1.3 MySQL为什么使用B+Tree数据结构做索引

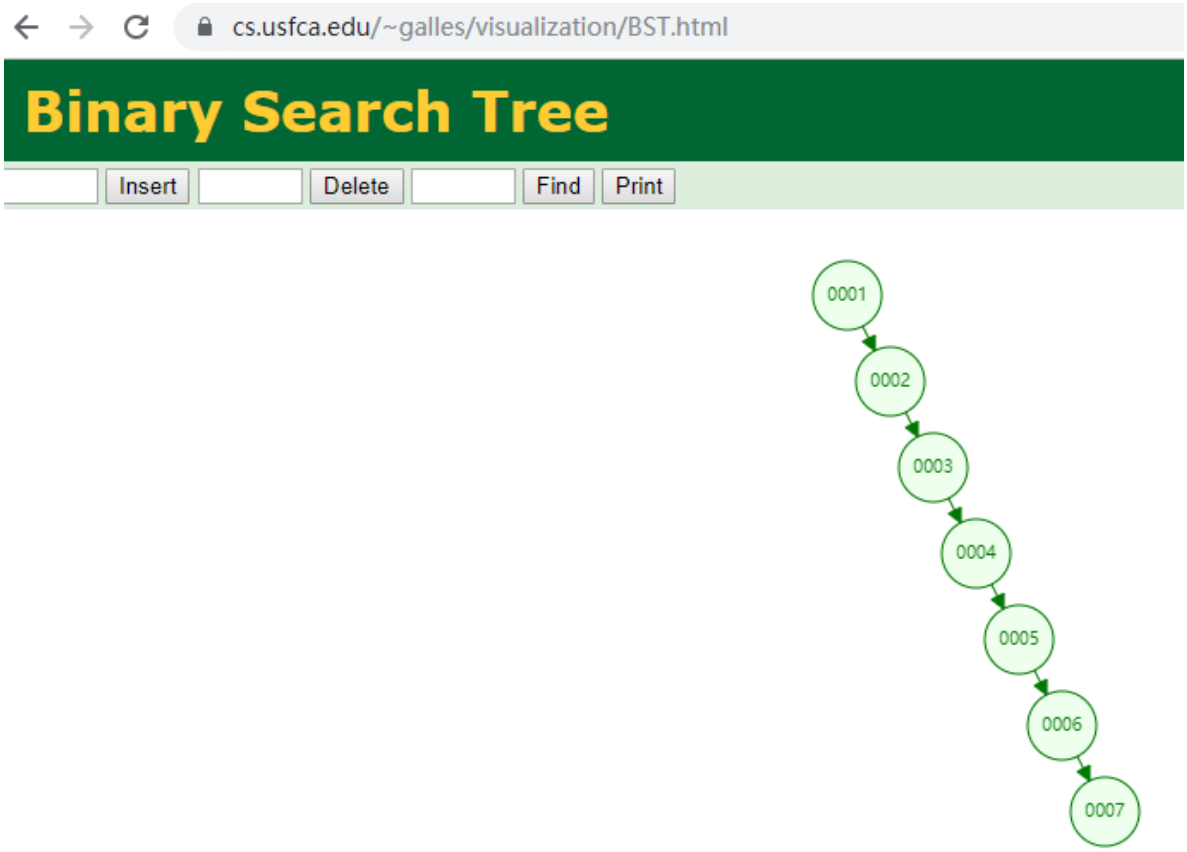
> 操作系统一次IO【一页】的数据量为4KB
>
> mysql一次IO的数据量为16KB
>
> 为何使用B+Tree的由来：从二叉树-->B+Tree的推演

1.3.1 二叉查找树 Binary Search Tree



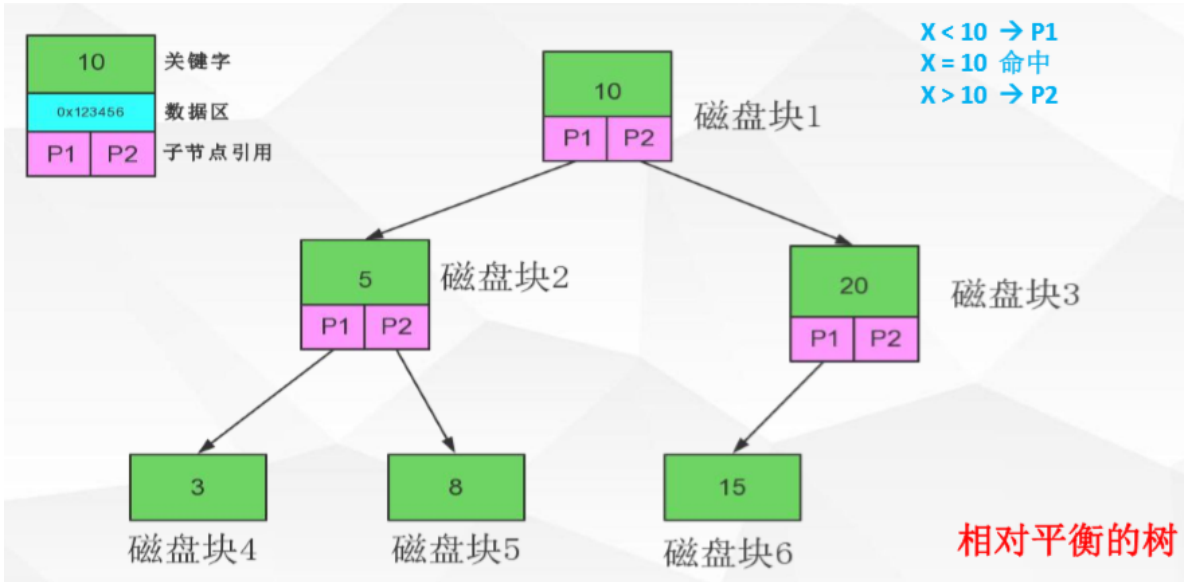
二叉树存在的弊端

> 当数据一次递增/减的方式插入的时候，二叉树会变为链状，使树失去平衡，导致查询节点过多。
>
> 二叉树演示地址



1.3.2 平衡二叉查找树 Balanced binary search tree

> - **平衡二叉查找树 又叫做 AVL树（相对平衡树）：其中某个节点的子节点的高度差不大于1。**
> - **平衡二叉查找树，解决了二叉树的树不平衡问题。**



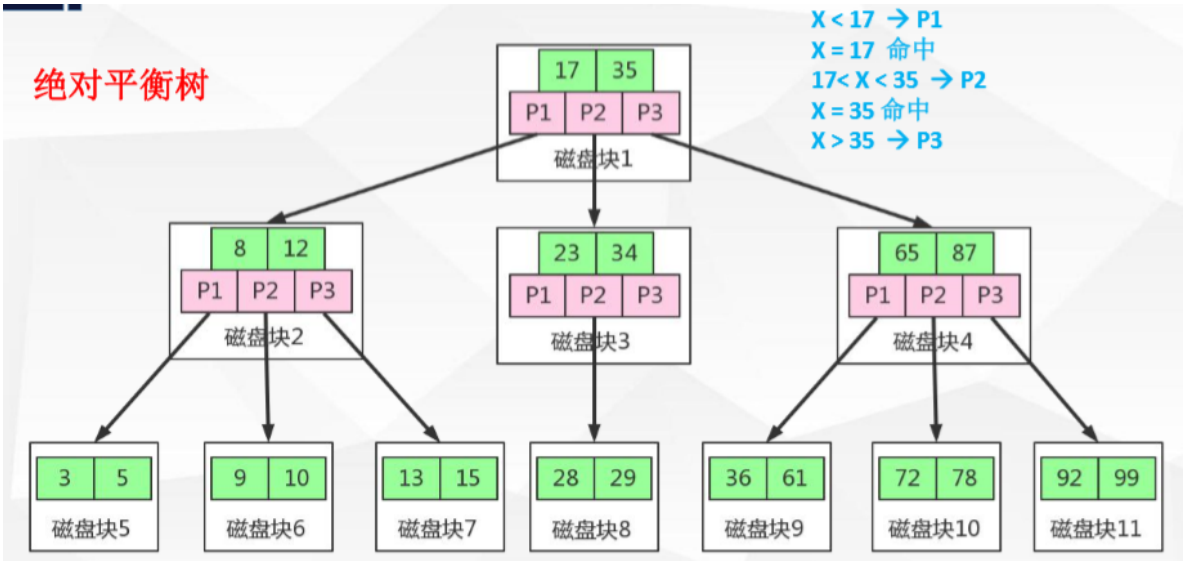
平衡二叉查找树缺点：

> - 它太深了
>   - 数据处的（高）深度决定着他的IO操作次数，IO操作耗时
> - 它太小了
>   - 每一个磁盘块（节点/页）保存的数据量太小了,没有很好的利用操作磁盘IO的数据交换特性。
>   - 也没有利用好磁盘IO的预读能力（空间局部性原理），从而带来频繁的IO操作

1.3.3 多路平衡查找树 B-Tree

- 保证了每个磁盘块可以存多个数据，使的整个树变得更矮更宽，保证一次IO能获取每层的节点数据会更多。
- 可以设定每个节点的路数和节点包含几个数据（路数-1）
- 17/35存储的数据结构如图

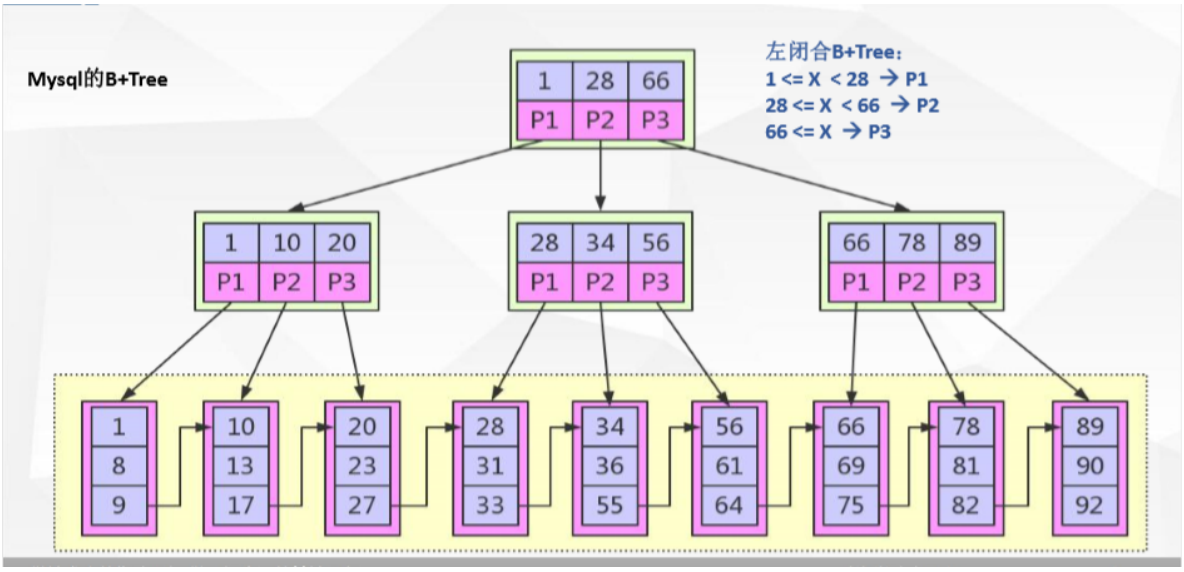
- P1/P2/P3，存储的是下一个叶子节点的引用。



多路平衡查找树 B-Tree的缺点：

- 由于每个节点都存储了每个数据17 35对应的完整数据区，占用的容量会很大，导致每次IO到的数据就会相对较少。
- 查询效率不稳定性：查询第一层数据可能需要0.001秒，查询第1000层叶子节点的数据可能花费的时间就是0.1秒。此种树导致查询效率不稳定。

1.3.4 加强版多路平衡查找树 B+Tree



B+Tree 与 B-Tree的区别

> 1，B+节点关键字搜索采用闭合区间
> 2，B+非叶节点不保存数据相关信息，只保存关键字和子节点的引用
> 3，B+关键字对应的数据保存在叶子节点中
> 4，B+叶子节点是顺序排列的，并且相邻节点具有顺序引用的关系

MySQL为什么选用B+Tree

> B+树是B-树的变种（PLUS版）多路绝对平衡查找树，他拥有B-树的优势
> B+树扫库、表能力更强
> B+树的磁盘读写能力更强
> B+树的排序能力更强
> B+树的查询效率更加稳定（查询第一层数据可能需要0.001秒，查询第1000层叶子节点的数据可能花费的时间就是0.1秒。）

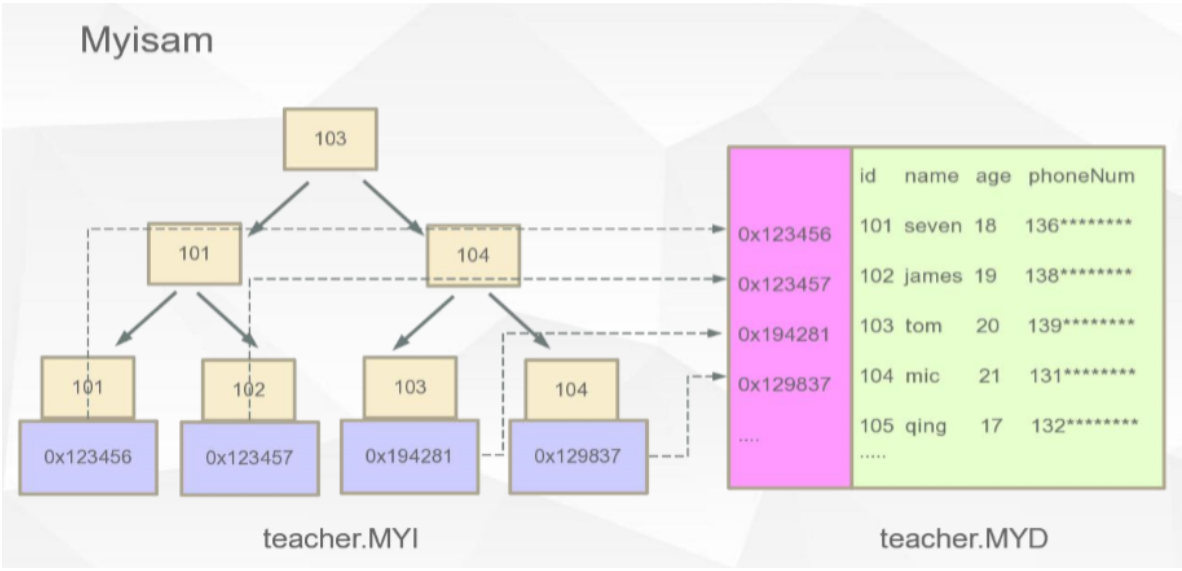
1.4 Mysql B+Tree索引体现形式

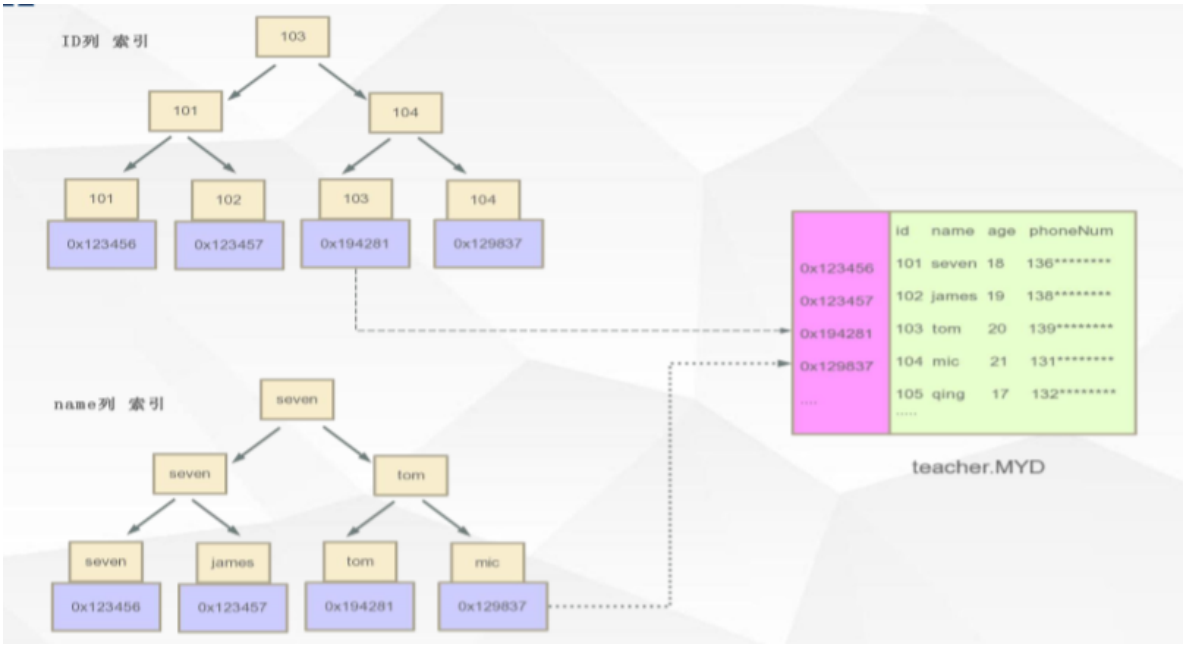
1.4.1 MYISAM引擎在mysql中B+Tree索引体现形式

MYISAM引擎，存储数据有三个文件：

> - tableName.frm:存储表定义文件
> - tableName.MYI：存储表的索引
> - tableName.MYD：存储表中数据

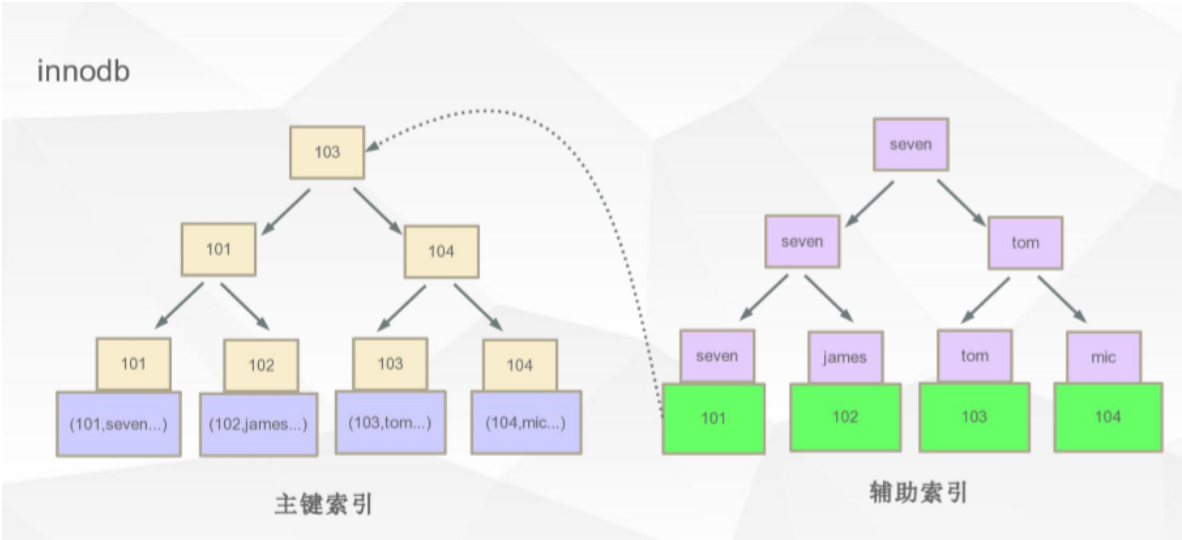
MYISAM引擎在B+Tree中，叶子节点存储的是数据的地址，具体数据还需要去teacher.myd文件中获取。

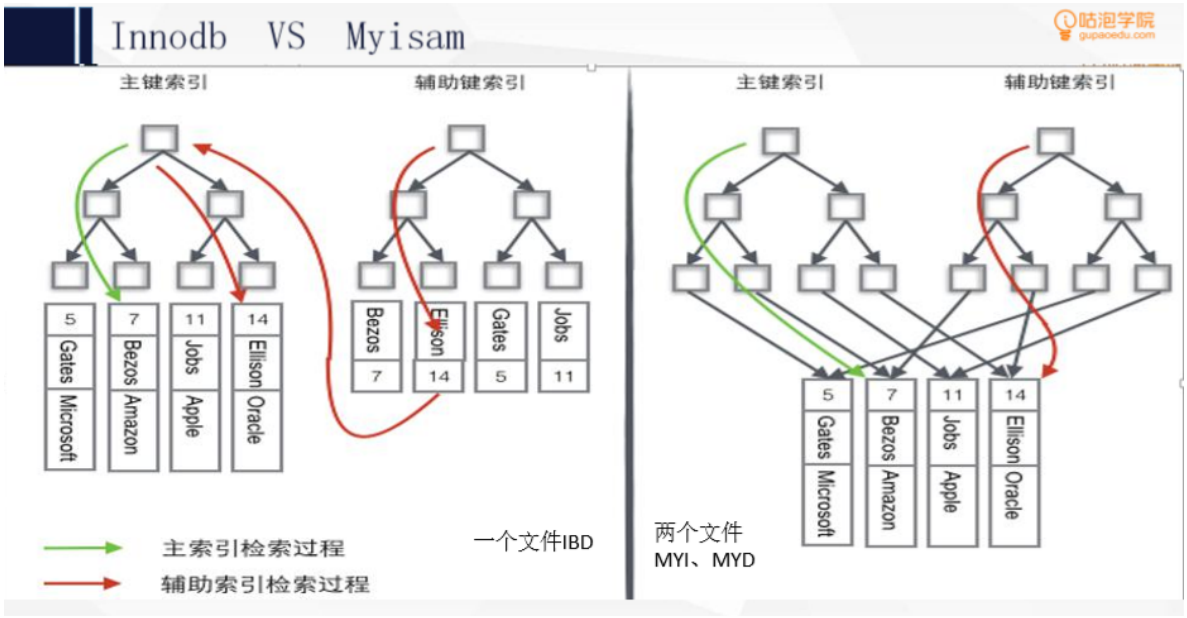




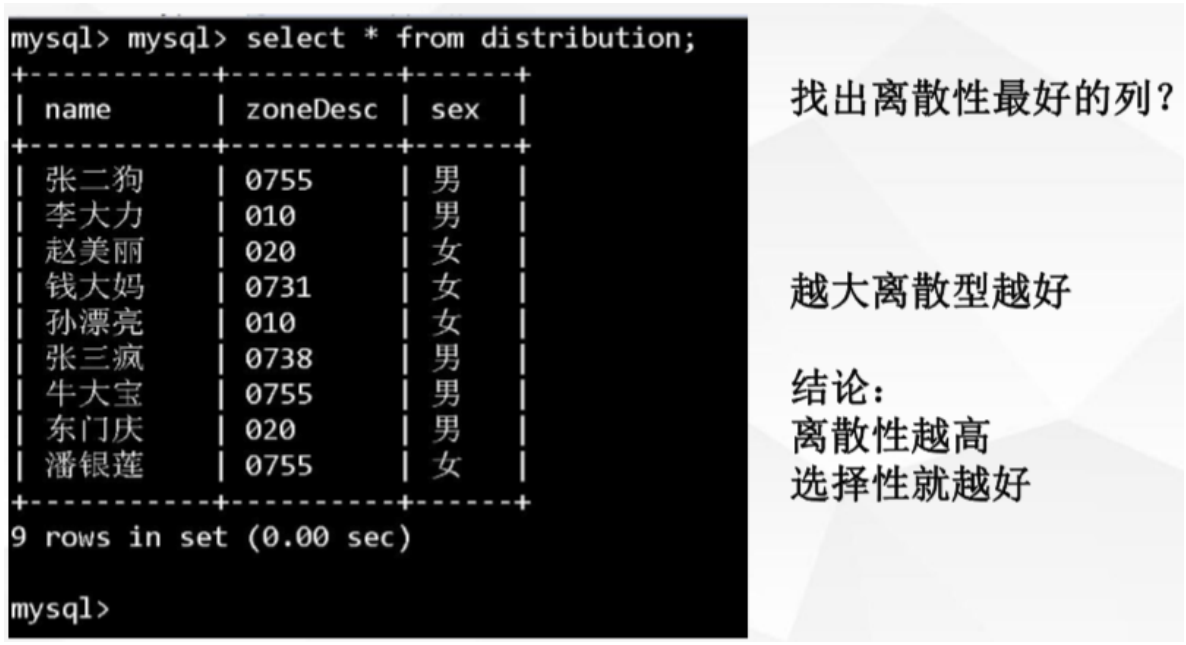
1.4.1 InnoDB引擎在mysql中B+Tree索引体现形式

> InnoDB认为，所有的索引最终都要指向主键ID索引
>
> 辅助索引：非ID索引的索引，最终的叶子节点存的是id，然后将id值去主键索引中查询最终的结果。



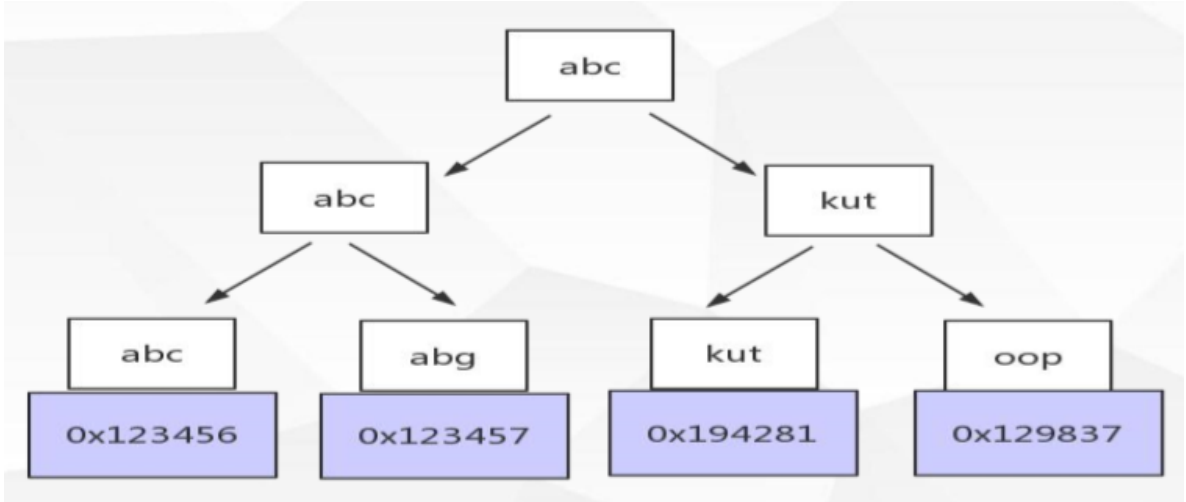


2.列的离散性



3.最左匹配原则

> 对索引中关键字进行计算（对比），一定是从左往右依次进行， 且不可跳过



4.联合索引

- 单列索引 节点中关键字[name]
  - 节点中关键字[name]
- 联合索引
  - 节点中关键字[name,phoneNum]
- 单列索引是特殊的联合索引（联合索引列选择原则 ）
  - 1，经常用的列优先 【最左匹配原则】
  - 2，选择性（离散度）高的列优先【离散度高原则】
  - 3，宽度小的列优先【最少空间原则】
  - 4，索引的列不允许有null存在

根据经常用的sql决定索引

经排查发现最常用的sql语句：

```
Select * from users where name = ? ;
Select * from users where name = ? and phoneNum = ?;
```

机灵的李二狗的解决方案：创建两个索引如下

```
create index idx_name on users(name);
create index idx_name_phoneNum on users(name,phoneNum);
```

错误，根据最左匹配原则，应该只建 联合索引即可（[name,phoneNum]）且name必须在前，为了照顾 where name=? 的sql语句

5.覆盖索引

> 如果查询列可通过索引节点中的关键字直接返回，则该索引称之为 覆盖索引。
> 覆盖索引可减少数据库IO，将随机IO变为顺序IO，可提高查询性能

6.总结

> **索引列的数据长度能少则少。**
> **索引一定不是越多越好，越全越好，一定是建合适的。**
> **匹配列前缀可用到索引 like 9999%，like %9999%、like %9999用不到索引； Where 条件中 not in 和 <>操作无法使用索引； 匹配范围值，order by 也可用到索引；**
> **多用指定列查询，只返回自己想到的数据列，少用select *；**
> **联合索引中如果不是按照索引最左列开始查找，无法使用索引；**
> **联合索引中精确匹配最左前列并范围匹配另外一列可以用到索引；**
> **联合索引中如果查询中有某个列的范围查询，则其右边的所有列都无法使用索引；**