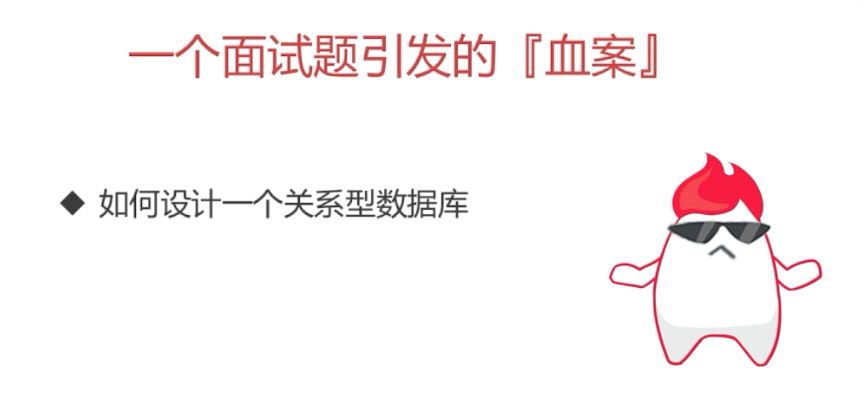
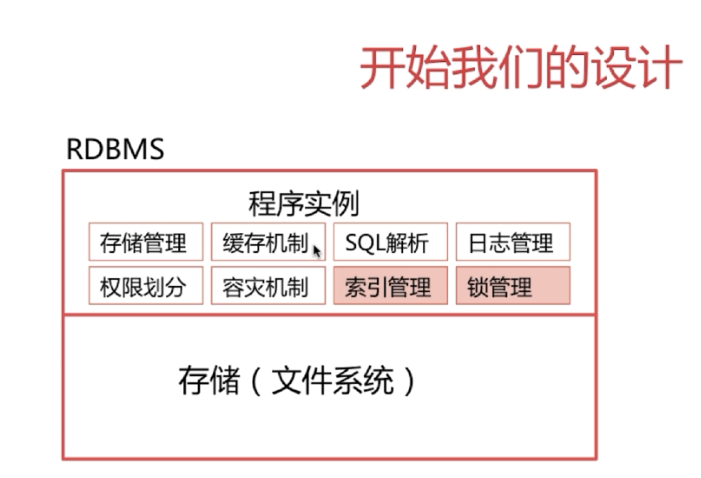
# 第3章

## 3-1数据库架构

### 如何设计一个关系型数据库





回答：首先将其划分为两个模块，一个是存储模块，该部分类似一个文件系统，将数据持久化磁盘中。另一个是程序实例模块，来对存储进行逻辑上的管理，将逻辑关系转换为物理存储关系的存储管理，优化执行效率的缓存模块，将sql语句进行解析的sql语句模块，记录操作的日志管理模块，进行多用户管。理的权限划分模块，灾难恢复模块，优化数据查询的索引模块，支持并发操作的锁管理

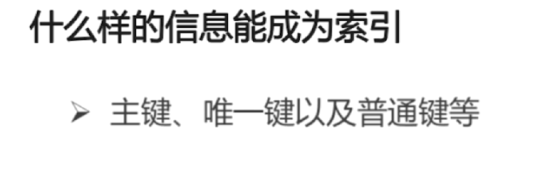
### 索引模块



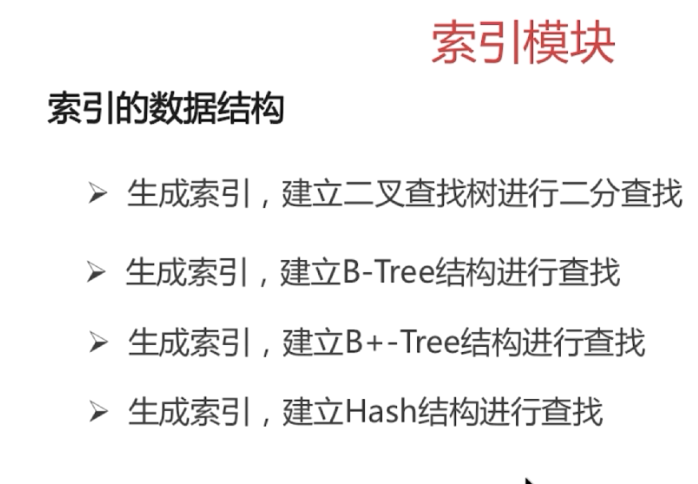
#### 为什么要使用索引

全表扫描效率慢，索引的灵感来自字典，使用索引可以快速查询信息

#### 什么样的信息能成为索引



#### 索引的数据结构（通常是b+树）



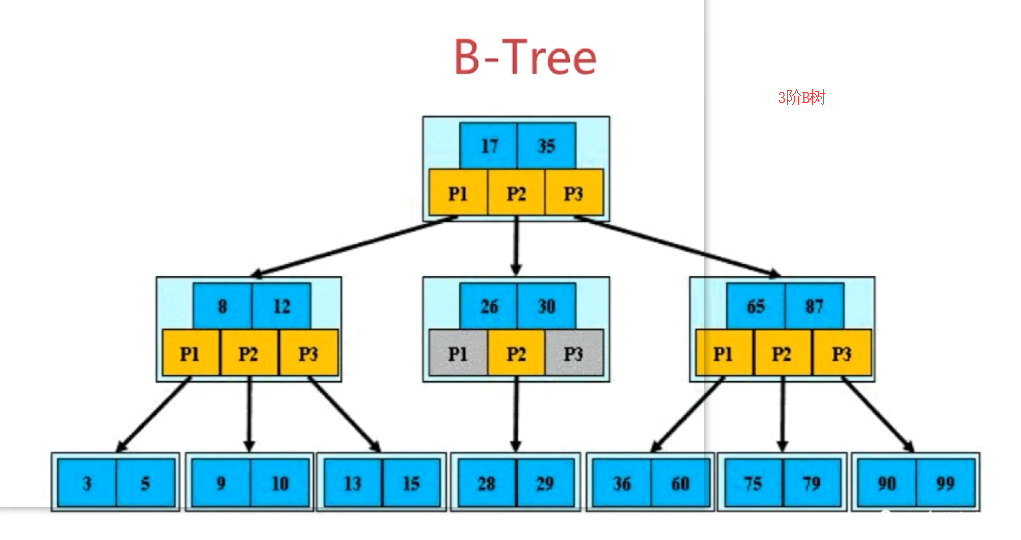
## 3-2优化你的索引-运用二叉查找树

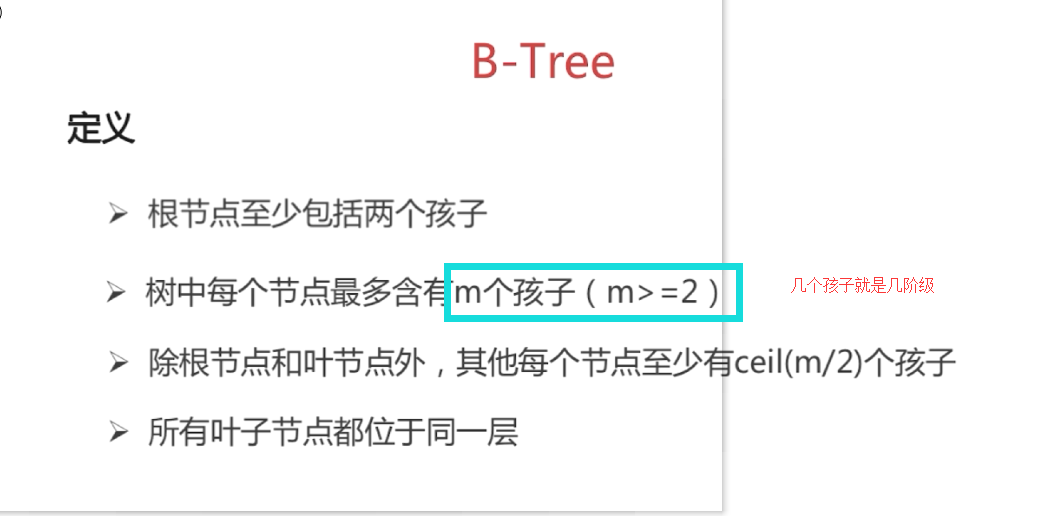
### 二叉查找树（对半搜索，时间复杂度log2(n),缺点1是变成线性查找树，时间复杂度为o(n)，缺点2是树的深度很深）

查找，既要提高时间复杂度，又要降低io的次数，（我们的想法的是每个节点存储的数据多一点，就想到Btree）

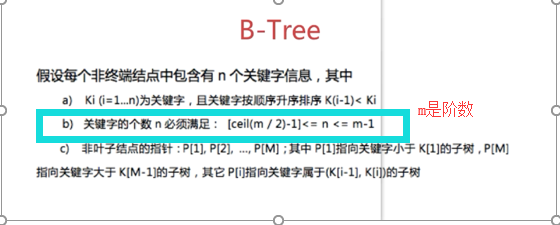
## 3-3优化你的索引-运用B树

### 优化你的索引-运用B树





除根节点和叶子节点，每个节点至少又ceil（m/2）个孩子，目的只有一个，就是让每隔索引块尽可能存储多的信息，让树的高度降低，减少io次数



B-tree 的实现

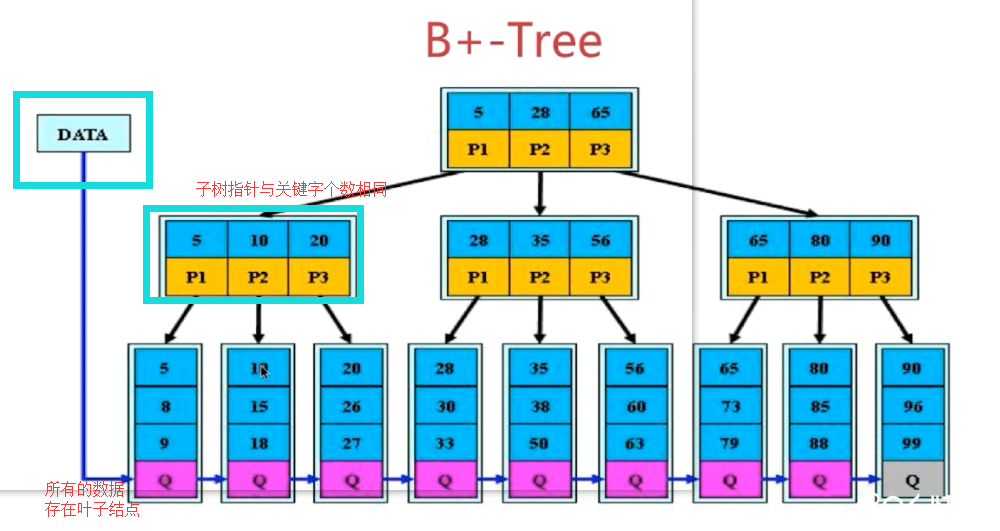
2-3树的插入实现的三种情况

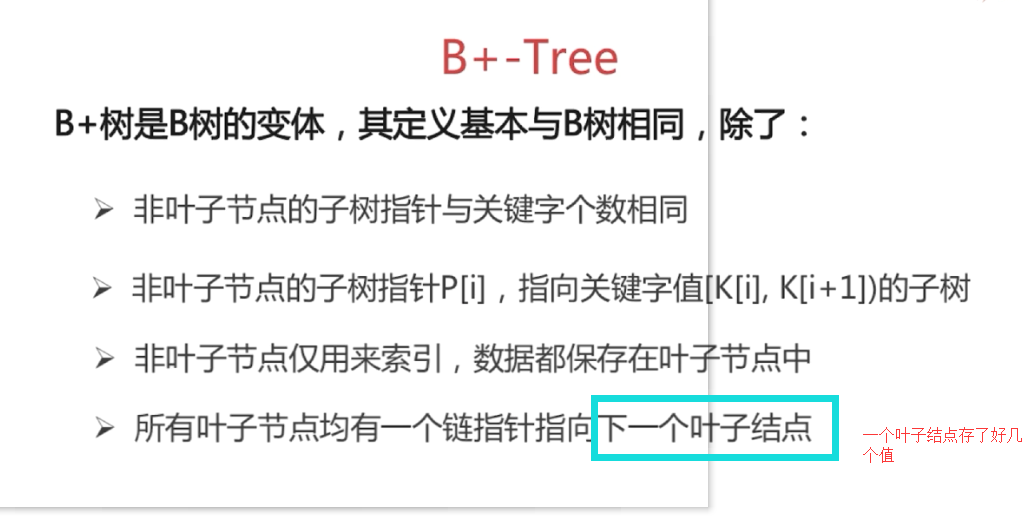
* 对于空树，插入一个值就行了，然后变成2节点
* 插入节点到一个只有一个值的2节点上，变成2个值的3节点就行了
* 插入到2个值的3节点上需要选择其一来上移

2-3树的删除实现

* 对于2个值的3节点，直接删除就行
* 对于一个值的2几点需要考虑4种情况（对于删除叶子节点是1值2节点的4种情况）
* 此节点的双亲也是2结点，且拥有一个3节点的右孩子，只要删除该节点后，在右旋转
* 此节点的双亲也是2结点，它的右孩子也是2结点，办法就是让右孩子变成3结点
* 双亲是3结点，等于将这个双亲结点拆分
* 满二叉树的情况：\*\*\*\*\*\*
* 如果删除的是非叶子结点且为1值2结点的话，通过将树按中序遍历后得到的前驱或者后继元素来补位。

## **3-4优化你的索引- 运用B+树**





结论：

B+树的磁盘读写代价更低

（B+树的内部结构没有指向关键字具体信息的指针（也就是说不存放数据只存放索引），其内部结点相对于B树更小，如果所有内部结点的关键字存放在同一盘块中，这个盘块所能容纳的关键字数量也越多，一次性读入内存中需要查找的关键字也越多，相对来说IO查找次数也就降低了）

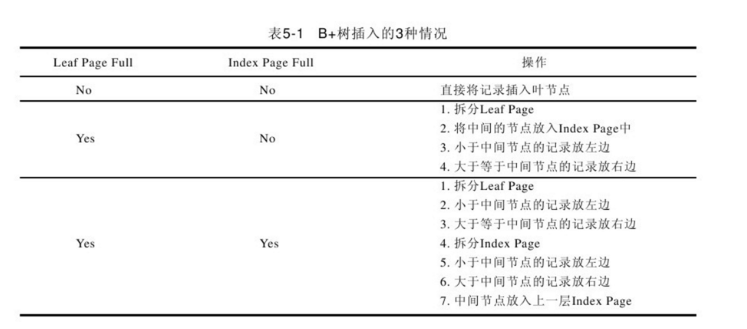
B+树的查找效率比较稳定

由于内部节点并不是指向最终文件的结点，而只是叶子结点关键字的索引，所以任何关键字的查找必须走从跟结点到叶子节点的路，所有关键字查询的长度相同

B+树更有利于对数据库的扫描（利用索引做范围查询）

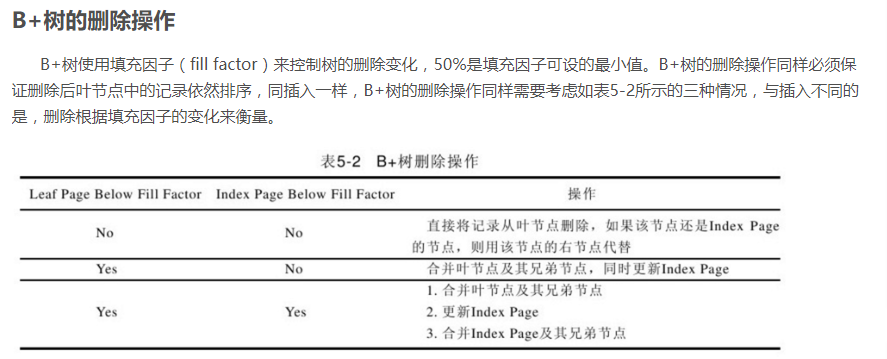
B树提高磁盘IO性能并没有解决元素遍历效率低下的问题，而B+树只需要遍历叶子结点，就可以全部扫描关键字信息

b+树插入



<https://blog.csdn.net/qq_35571554/article/details/82760568>

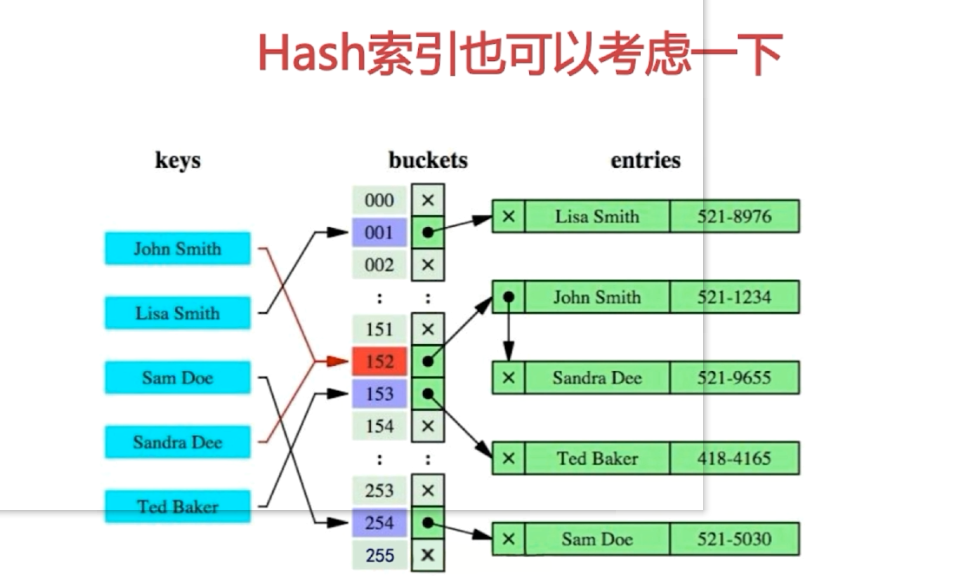
b+树删除



<https://blog.csdn.net/sunshine_lyn/article/details/82747596>

## 3-5优化你的索引-运用hash以及bitmap

### hash索引



理论上hash索引的查询效率比B+树高，为什么我们不用hash索引取代b+树索引呢

hash索引的缺点

buckets存放两个东西（一个行指针，一个数据）

* 仅仅能满足”=”,“in”,不能使用范围查询（hash索引是存放的经过hash运算的值

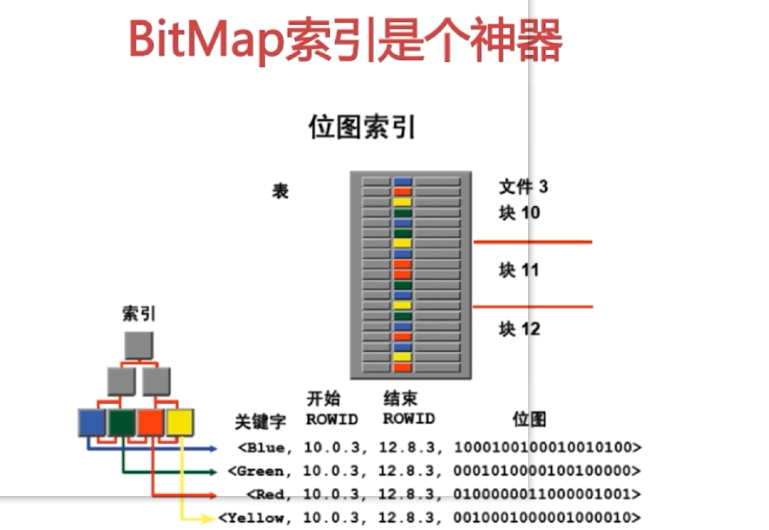
）

* 无法被用来避免数据的排序操作（hash索引是存放的经过hash运算的值

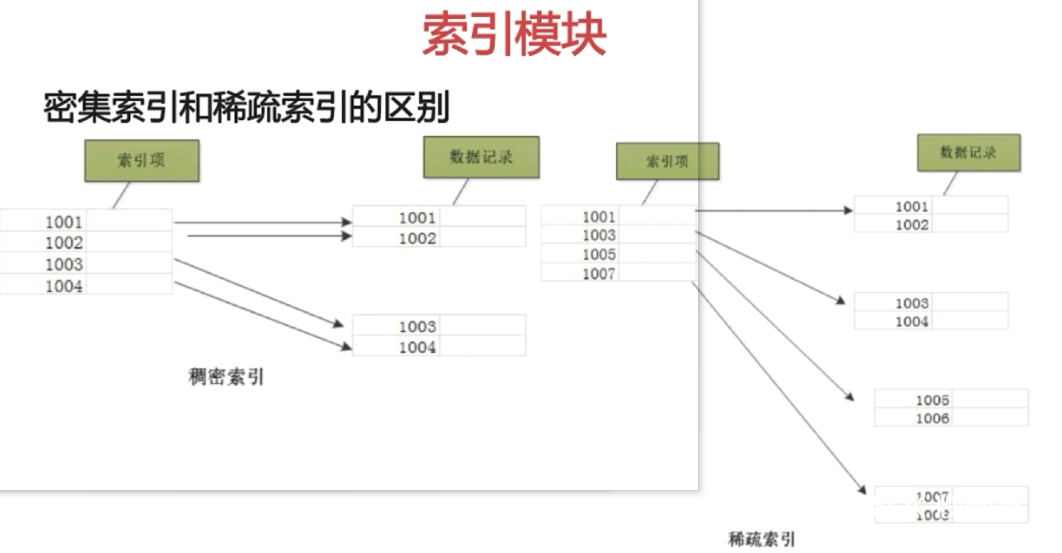
）

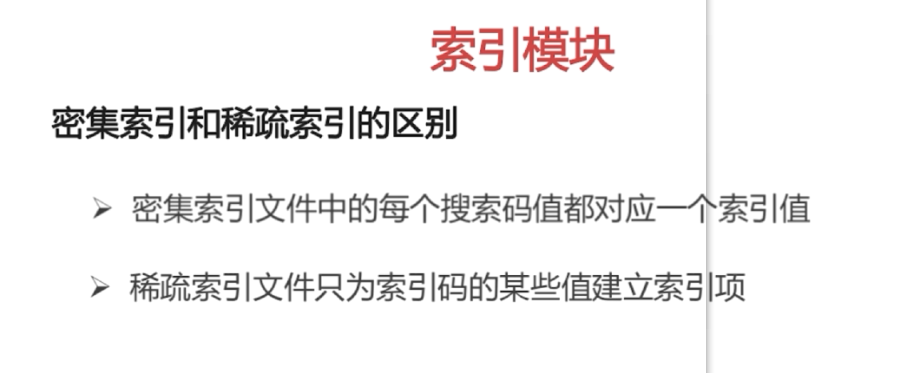
* 不能利用部分索引键去查询（对于组合索引，hash是对组合键合起来计算，不是单独去对每个键来计算，而B+支持组合索引的部分扫描）
* 不能避免表扫描（hash值存放在一个buckets中，由于不同索引存在相同的hash值，即使取出满足某个hash键值的数据，无法从hash索引中直接完成查询，需要访问buckets中的实际数据）
* 遇到大量hash值相等的情况后性能并不一定就会比B-Tree索引高

### bitmap索引是个神器（适用于某个字段的值只有固定的几种情况，不适合高并发的联机处理系统）

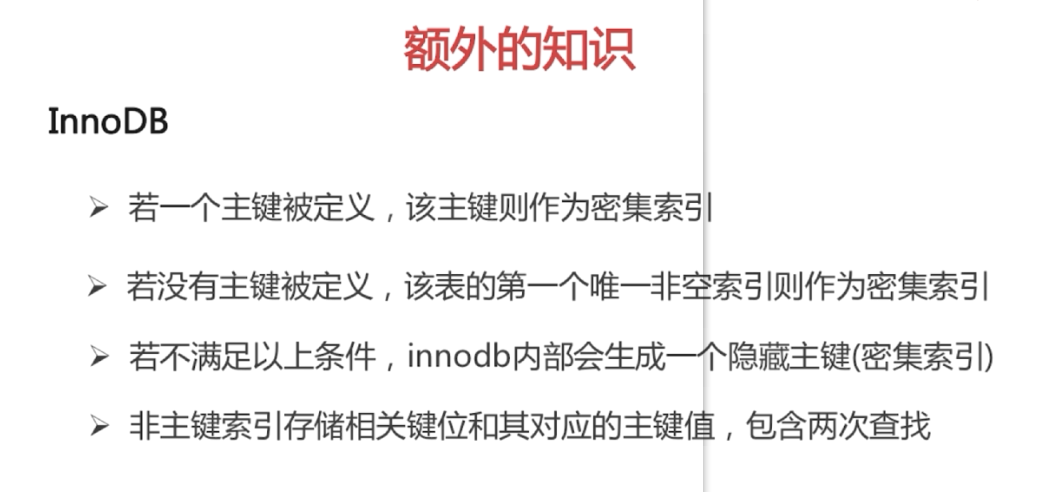


## 3-6密集索引和稀疏索引的区别

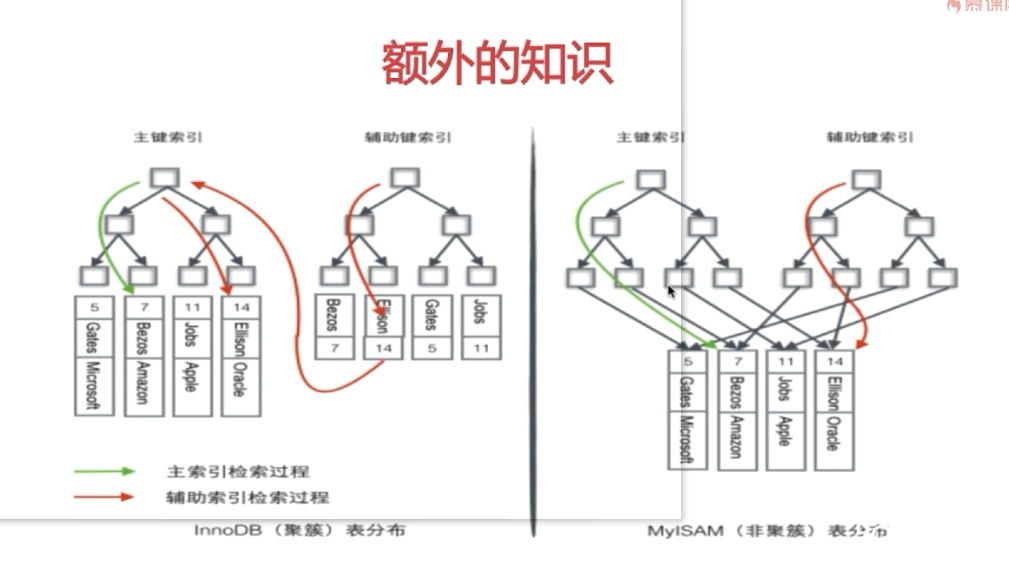


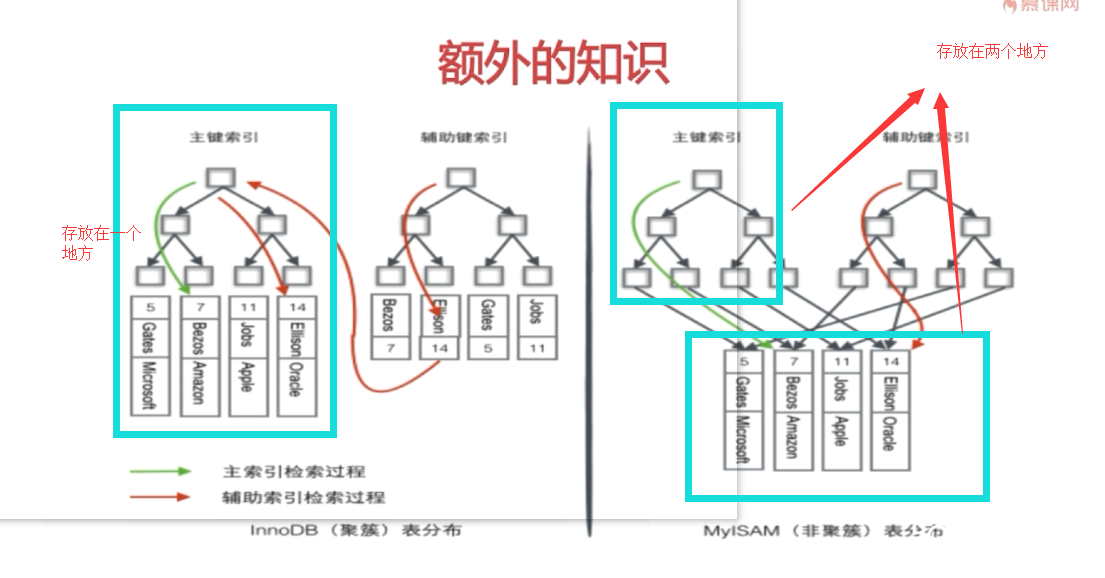


### InnoDB

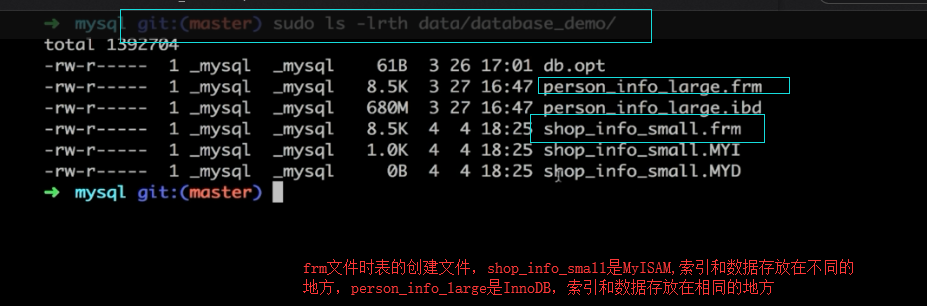


看下图：非主键索引就是先根据辅助键找到对应的主键值，然后在根据 主键值去找对应的值（下图的红线标注的过程）





查看数据库



## 3-7索引额外的问题值如何调优sql

为什么要使用索引 避免全表扫描去查找数据，提升检索效率

什么样的信息能成为索引 主键，唯一键，让数据具有区分的字段

索引的数据结构 主流是b+树，还有hash和bitmap，mysql数据库不支持bitmap

密集索引和稀疏索引的区别 密集索引文件中的每隔搜索码值都对应一个索引值，稀疏索引文件只为索引码的某些值简历索引项

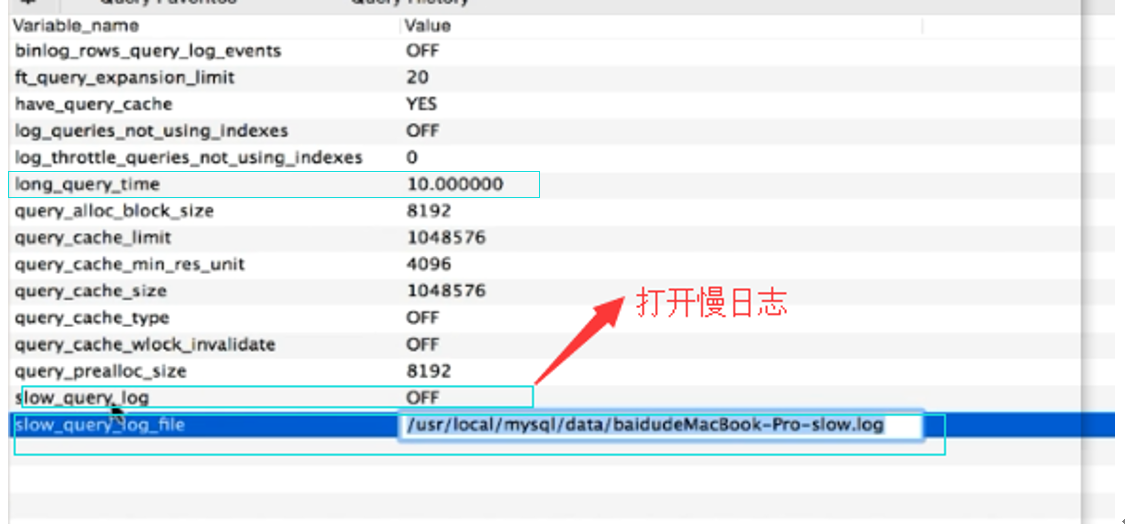
### 索引模块衍生出来的问题，以mysql为例

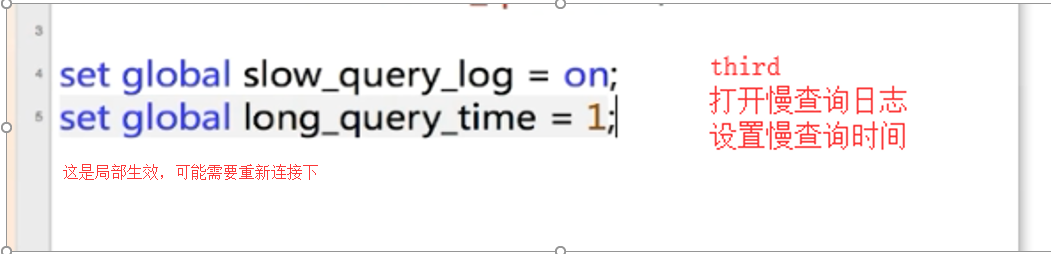
#### 如何定位并优化查询sql（步骤1，2，3）

1. 根据慢日志定位慢查询sql

慢日志就是用来记录执行比较慢的sql

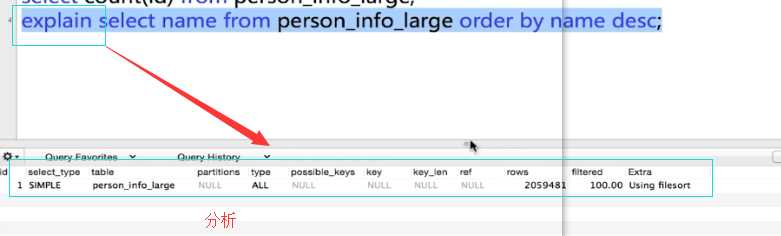
* show variables like ‘%quer%’ 模糊查询，显示Mysql服务器配置信息



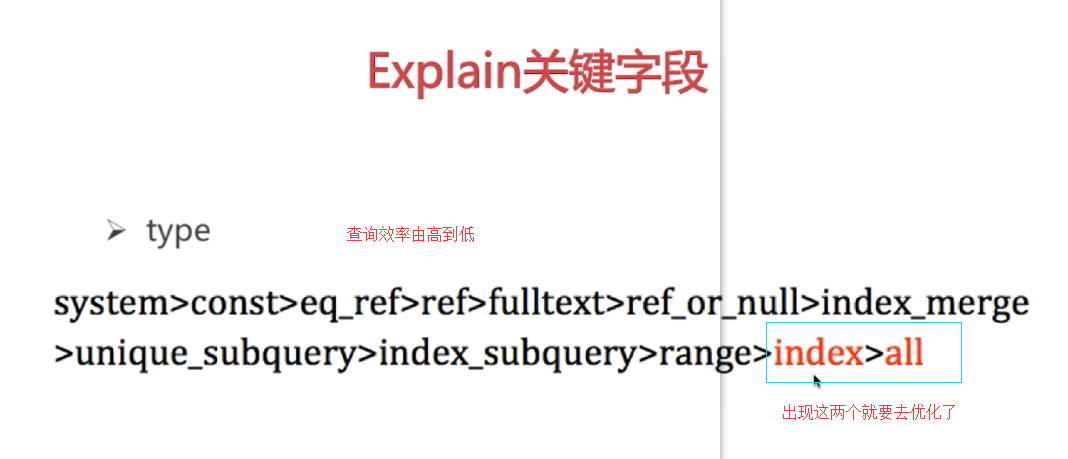
 

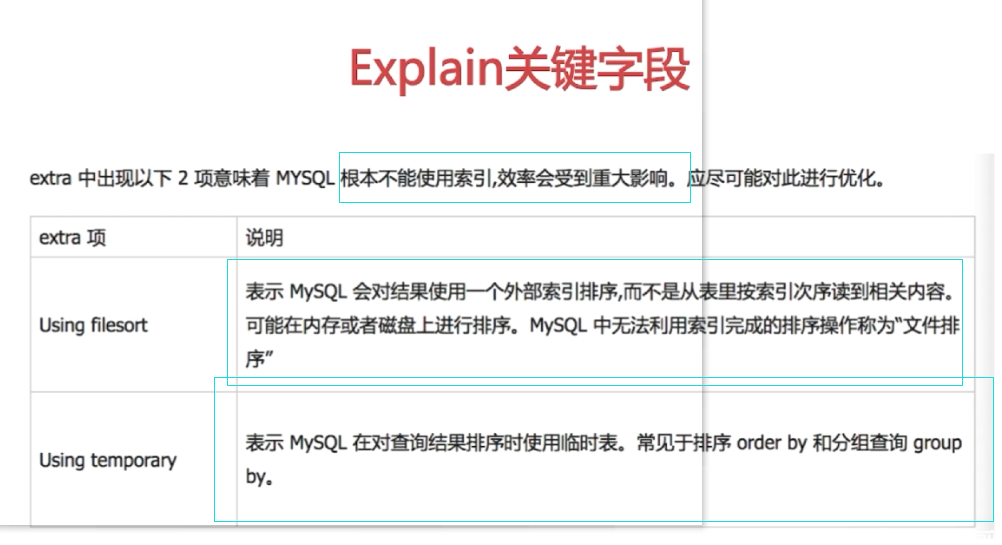
注意，慢查询记录只在本次会话中有效，一但关闭会话，就会置为0

1. 使用explain等工具分析sql



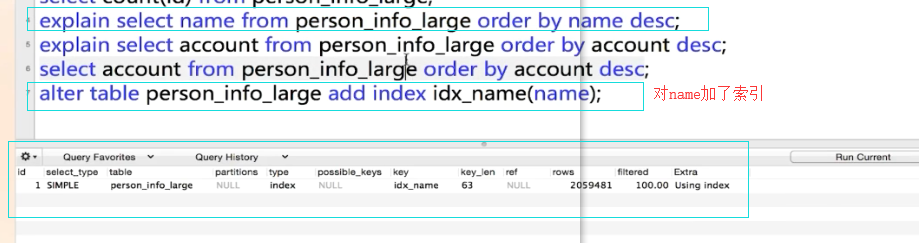
type表示mysql找到需要数据行的方式

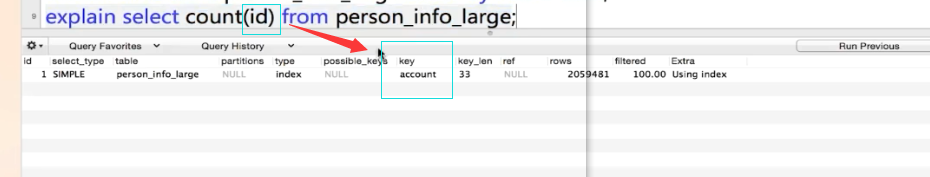




mysql中无法利用索引进行的排序，我们统称为文件排序。（这个排序可能在磁盘，可能在内存）

1. 修改sql或者尽量让sql走索引

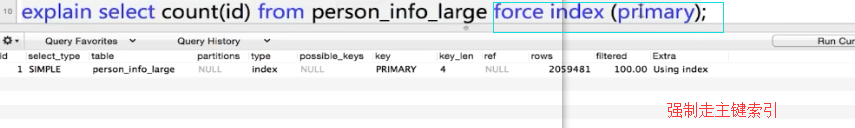




对于上面的查询语句走的是account索引没有走id索引的原因。

mysql的查询优化器做决定，最重要的目标是使用索引，使用最严格的索引来消除尽可能多的数据行，（最终目标是提交select语句查找数据行，而不是排除数据行），优化器排除数据行的原因在于排除数据行的速度越快，找到与条件匹配的数据行就越快，因此查询优化其会根据它的分析和判断标准决定走那个索引。这里没有走主键索引的原因是，

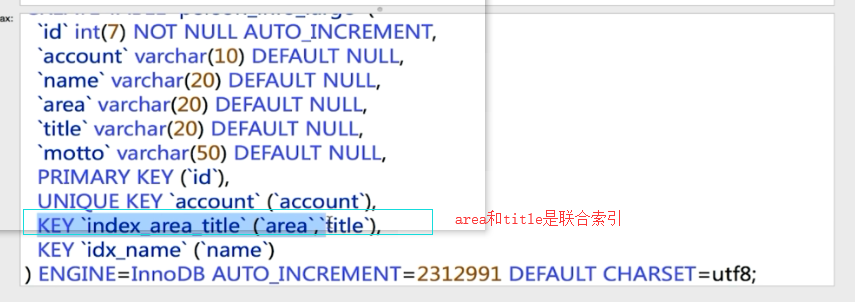
密集索引的叶子结点把其他列的数据也存放在叶子结点中，效率比稀疏索引低，因为稀疏索引只存了关键字以及主键的值，这样在内存中可以加载更多的关键字和值



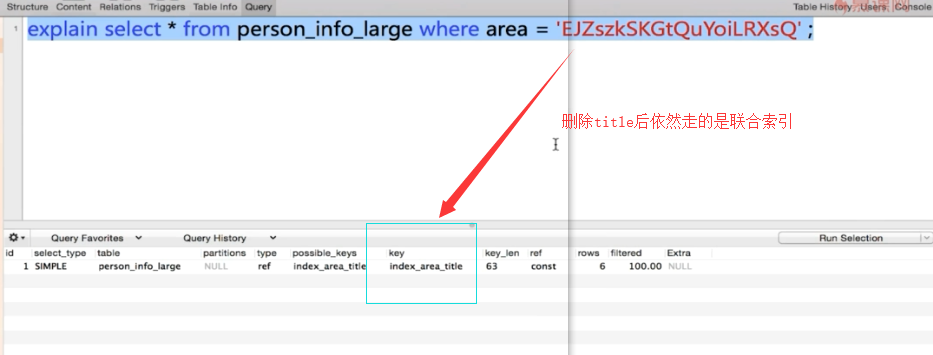
#### 联合索引的最左匹配原则的成因

#### 索引是建立的越多越好吗

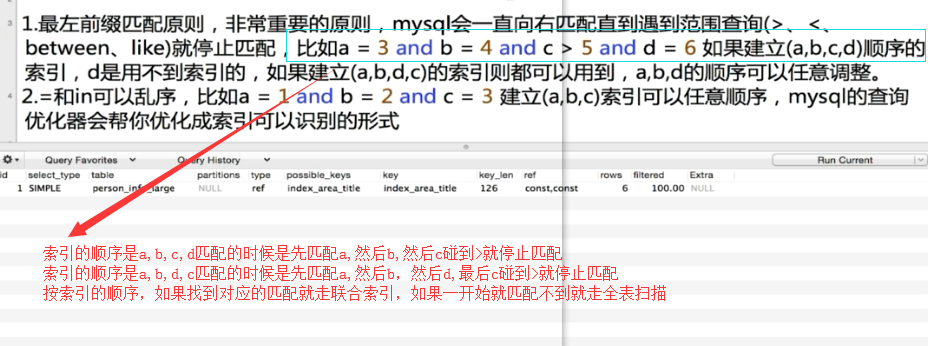
## 3-8联合索引的最左匹配原则的成因





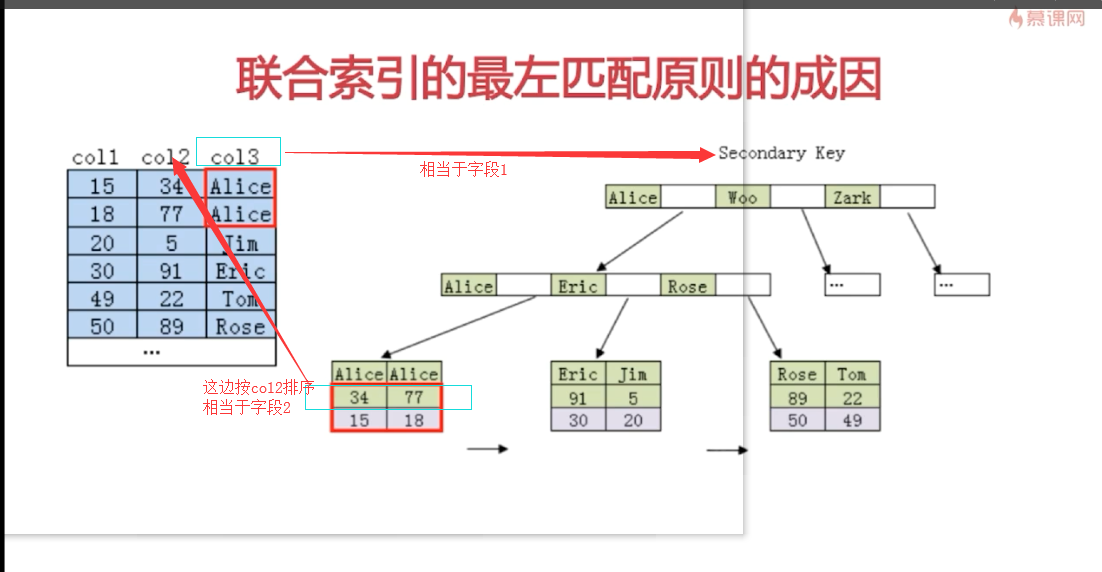




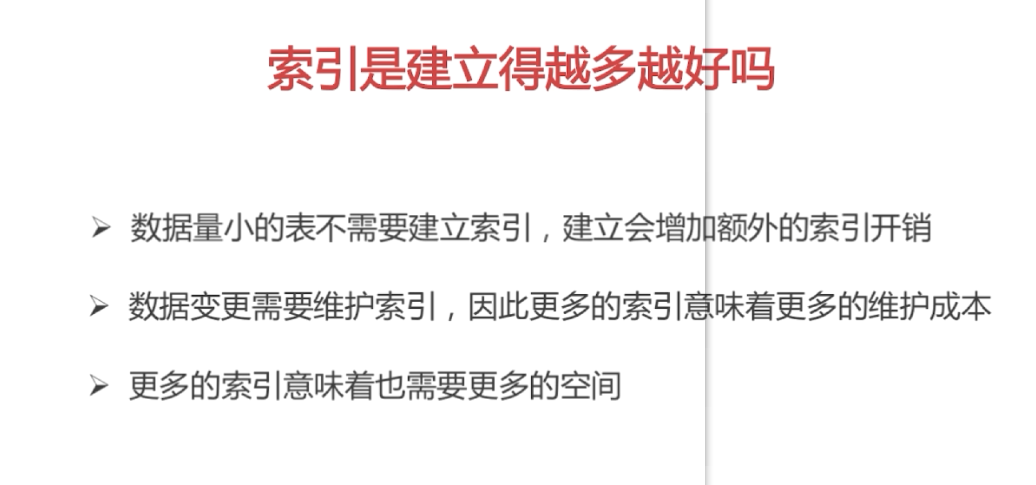


### 联合索引的最左匹配原则的成因

mysql会对联合索引的第一个索引字段1排序，在第一个排序的基础上在对第二个索引字段2排序，其实类似实现了order by 字段1，order by 字段2这种排序规则，所以第一个字段是绝对有序，第二个字段就是无序的，因此使用第二个字段进行条件判断是用不到索引的



## 3-9索引额外问题之索引是建立越多越好吗



索引总结

复习索引相关的知识，学习主外键，唯一键，研究mysql的innodb和myASIM