## 1.既然你提到InnoDB使用的B+ 树的索引模型，那么你知道为什么采用B+ 树吗？这和Hash索引比较起来有什么优缺点吗？

**A：**（突然觉得这道题有点难，但是我还是凭借着自己的知识储备简单的回答上一些）因为Hash索引底层是哈希表，哈希表是一种以key-value存储数据的结构，所以多个数据在存储关系上是完全没有任何顺序关系的，所以，对于区间查询是无法直接通过索引查询的，就需要全表扫描。**所以，哈希索引只适用于等值查询的场景。**而B+ 树是一种多路平衡查询树，所以他的节点是天然有序的（左子节点小于父节点、父节点小于右子节点），所以对于范围查询的时候不需要做全表扫描

B+ Tree索引和Hash索引区别？

哈希索引适合等值查询，但是无法进行范围查询

哈希索引没办法利用索引完成排序

哈希索引不支持多列联合索引的最左匹配规则

如果有大量重复的情况下，哈希索引的效率会很低，因为存在哈希碰撞问题

## 联合索引的最左匹配原则

当建立了联合索引时，查询时会遵循最左原则。即查询时，条件得包括联合索引中最左的列索引，比如用户表建立一个联合索引，adc，a是名字，b是年龄，c是性别。在用户表查询时的条件必须得有名字，才能走联合索引。因为b-tree索引是一种多路搜索树结构，它是根据联合索引中从左到右，建立索引结构的。如果没有name，从左到右进行搜索时，就不知道从哪个地方开始搜索， 就会走全表扫描。索引就失效了。这就是索引的最左匹配原则。

为什么要建立联合索引？

有1000W条数据的表，有如下sql:select \* from table where a = 1 and b =2 and c = 3,假设每个条件可以筛选出10%的数据，如果只有单值索引，那么通过该索引能筛选出1000W\*10%=100w 条数据，然后再回表从100w条数据中找到符合b=2 and c= 3的数据，然后再排序，再分页；如果是复合索引，通过索引筛选出1000w \*10% \*10% \*10%=1w，然后再排序、分页，哪个更高效，一眼便知

## 3.聚簇索引

**Q：刚刚我们聊到B+ Tree ，那你知道B+ Tree的叶子节点都可以存哪些东西吗？**

**A：**InnoDB的B+ Tree可能存储的是整行数据，也有可能是主键的值

**Q：那这两者有什么区别吗？**

**A：**（当他问我叶子节点的时候，其实我就猜到他可能要问我聚簇索引和非聚簇索引了）在 InnoDB 里，索引B+ Tree的叶子节点存储了整行数据的是主键索引，也被称之为聚簇索引。而索引B+ Tree的叶子节点存储了主键的值的是非主键索引，也被称之为非聚簇索引

**Q：那么，聚簇索引和非聚簇索引，在查询数据的时候有区别吗？**

**A：**聚簇索引查询会更快？

**Q：为什么呢？**

**A：**因为主键索引树的叶子节点直接就是我们要查询的整行数据了。而非主键索引的叶子节点是主键的值，查到主键的值以后，还需要再通过主键的值再进行一次查询

**Q：刚刚你提到主键索引查询只会查一次，而非主键索引需要回表查询多次。（后来我才知道，原来这个过程叫做回表）是所有情况都是这样的吗？非主键索引一定会查询多次吗？**

**A：**（额、这个问题我回答的不好，后来我自己查资料才知道，通过**覆盖索引**也可以只查询一次）

覆盖索引？

覆盖索引（covering index）指一个查询语句的执行只用从索引中就能够取得，不必从数据表中读取。也可以称之为实现了索引覆盖。

当一条查询语句符合覆盖索引条件时，MySQL只需要通过索引就可以返回查询所需要的数据，这样避免了查到索引后再返回表操作，减少I/O提高效率。

如，表covering\_index\_sample中有一个普通索引 idx\_key1\_key2(key1,key2)。

当我们通过SQL语句：select key2 from covering\_index\_sample where key1 = 'keytest';的时候，就可以通过覆盖索引查询，无需回表。

**Q：你们创建的那么多索引，到底有没有生效，或者说你们的SQL语句有没有使用索引查询你们有统计过吗？**

**A：**这个还没有统计过，除非遇到慢SQL的时候我们才会去排查

**Q：那排查的时候，有什么手段可以知道有没有走索引查询呢？**

**A：**可以通过explain查看sql语句的执行计划，通过执行计划来分析索引使用情况

## 4索引失效的情况？



### 4.1查询条件使用了where or。

### 如果条件中有or，即使其中有部分条件带索引也不会使用。只要多个条件中有一个不带索引，此条语句的索引都会失效。

注意：要想使用or，又想让索引生效，只能将or条件中的每个列都加上索引

### 4.2.like条件

时以 %号开头，会进行全表扫描。索引会失效

### 4.3联合索引中，

没有遵循最左匹配原则。

## 5.什么情况下不推荐使用索引？

## **什么情况下不推荐使用索引？**

### 5.1) 数据唯一性差

（一个字段的取值只有几种时）的字段不要使用索引

比如性别，只有两种可能数据。意味着索引的二叉树级别少，多是平级。这样的二叉树查找无异于全表扫描。

### 5.2) 频繁更新的字段不要使用索引

比如logincount登录次数，频繁变化导致索引也频繁变化，增大数据库工作量，降低效率。

### 5.3字段不在where语句出现时

不要添加索引,如果where后含IS NULL /IS NOT NULL/ like ‘%输入符%’等条件，不建议使用索引

只有在where语句出现，mysql才会去使用索引

### 5.4where 子句里对索引列使用不等于（<>），

使用索引效果一般