索引

1. 介绍

索引是一种可以帮助我们高效获取数据的数据结构。

索引出现的目的：提高查询的效率。

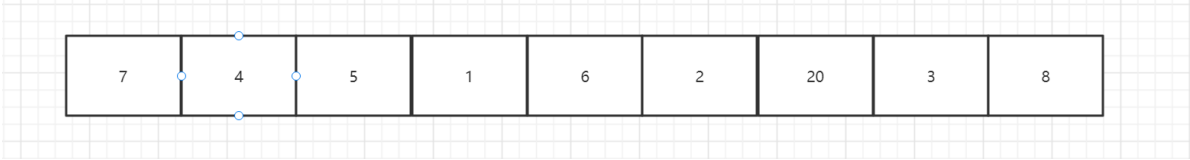
索引就类似于我们字典开头的目录，可以帮助我们更加快速的查找到对应的内容。

2. 数据结构

什么样的数据结构可以帮助我们快速的找到对应的数据呢？

- 查询单个值
- 查询范围值

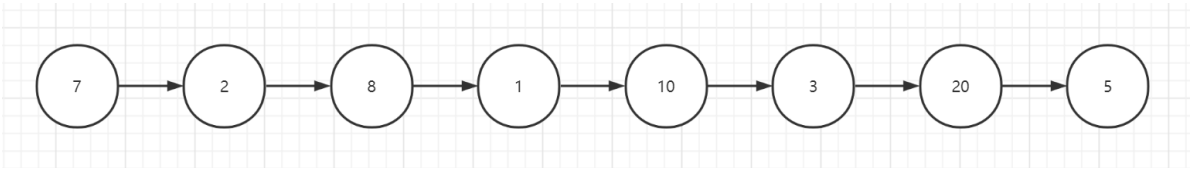
数组



- 查询单个值：遍历
- 查询范围值：遍历

不能帮助我们提高查询的效率。

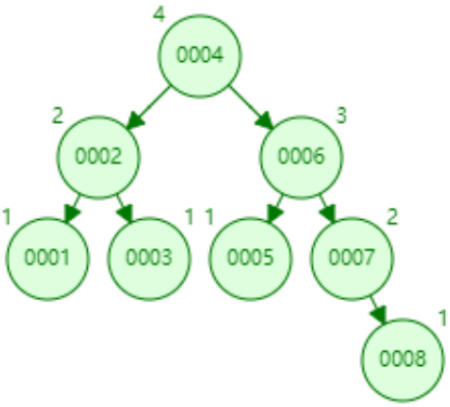
链表



- 查询单个值：遍历
- 查询范围值：遍历

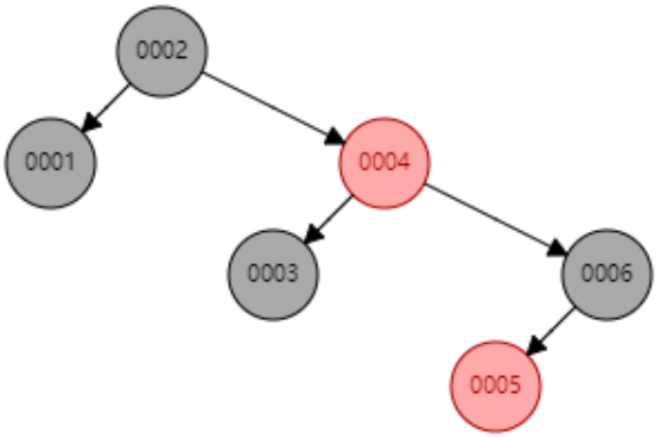
不能帮助我们提高查询的效率。和数组是类似的。

二叉树



- 查询单个值：对比遍历，有明显的提升
- 查询范围值：对比遍历来说，效率稍微高一点，但是依然不够方便

红黑树

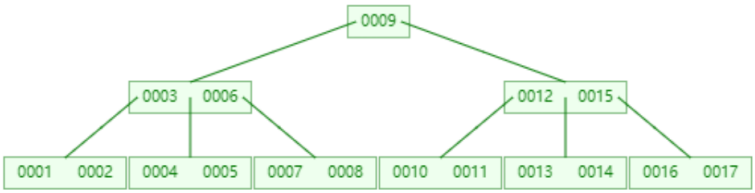


- 查询单个值：对比遍历，有明显的提升
- 查询范围值：和二叉树类似，效率稍微比遍历高一点，但是依然不够方便

> 其实，红黑树和二叉树存在类似的问题：
>
> 1. 当数据量很大的时候，树的高度会比较高
> 2. 效率查找不够方便

B-树

> 针对于以上的二叉树和红黑树，只有两个叉的情况，B-树是一种多路搜索树，存在多个叉。所以同样的数据量，B-树的高度应该比二叉树和红黑树要低很多。



- 查询单个值：对比遍历要方便一些；对比红黑树和二叉树因为树的高度更低，所以效率更高
- 查询范围值：也要在父子节点之间来回查找，不是很方便（对比遍历要方便一点）

> 为什么B树的高度降低之后，会比二叉树和红黑树查询的效率要更高？

> 因为磁盘的读取不是按需读取，而是会预读一部分内容，通常来说预读的大小是一个磁盘页（4k）的整数倍，通常情况下操作系统会读取一个内存页（16k）的大小；所以MySQL的设计者在考虑到这一点之后，就把每一个节点的大小设置为16k，所以树的高度越低，磁盘IO的次数就越少（每一层就是一次磁盘IO），读取的效率也就更高。

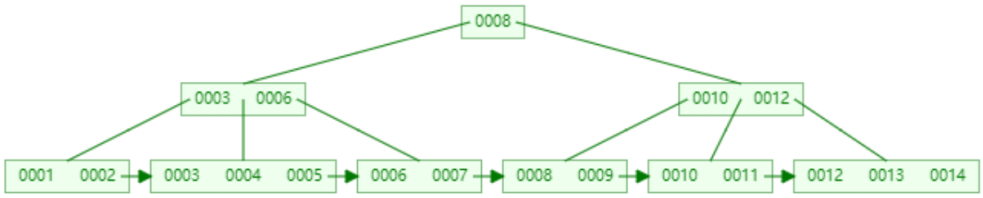
B+树

MySQL中真正使用的B+树是一种B树的变种，是在B树的基础之上进行了优化：

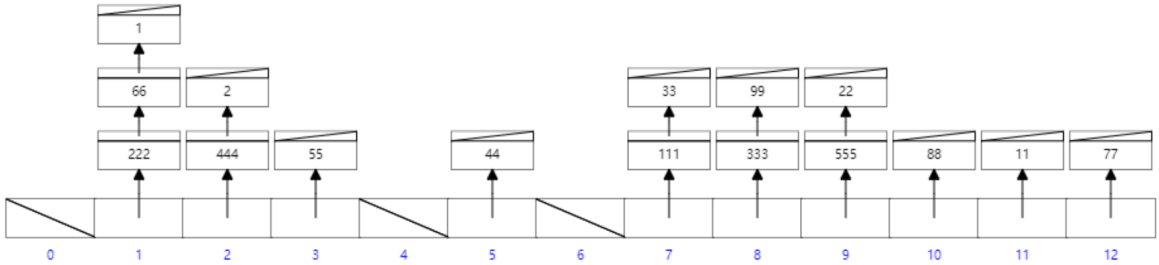
- **在所有的叶子节点之间维护一个指针，指向下一个叶子节点**
- **所有的非叶子节点在叶子节点中都冗余一份**

  （以上的两点解决了范围查找不方便的问题）
- **所有的非叶子节点，只存储key，不存储这个key对应的data；所有的data都在叶子节点中去存储**

  （这个优化可以有效的去降低树的高度，提高查询的效率）



Hash表



- 查询单个值：效率很高，很方便
- 查询范围值：不方便，需要遍历

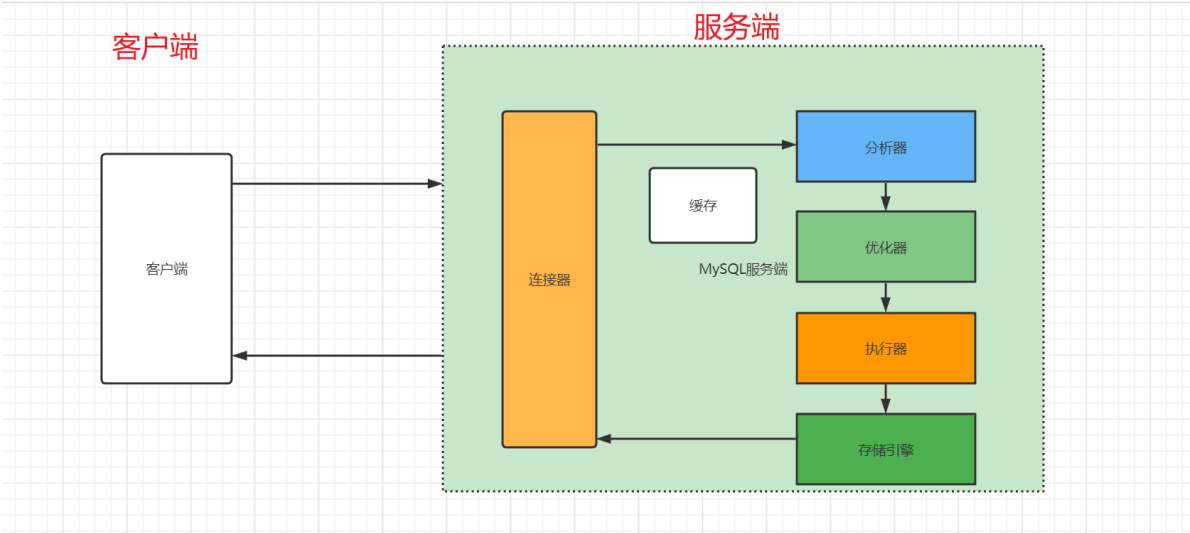
MySQL里面有使用Hash表来当做索引的情况，这种索引叫做Hash索引。一般来说，**Hash索引不推荐给用户自己使用**，因为MySQL不能确保用户使用Hash索引的表没有范围查找的需求。

3. 存储引擎

那么接下来的问题，就变成了B+树这种数据结构到底是怎样存储数据的？

换句话来说，其实就是MySQL表中的数据到底是怎么样存储到磁盘上的。

3.1 MySQL的组成结构



- 连接器：复制数据库连接的管理、权限的验证
- 分析器：词法分析、语法分析
- 优化器：优化SQL语句的性能，例如连接的先后顺序、索引的选择
- 执行器：执行SQL语句编译之后的命令，调用存储引擎的接口，操作数据
- 存储引擎：MySQL中真正用来存储数据的组件

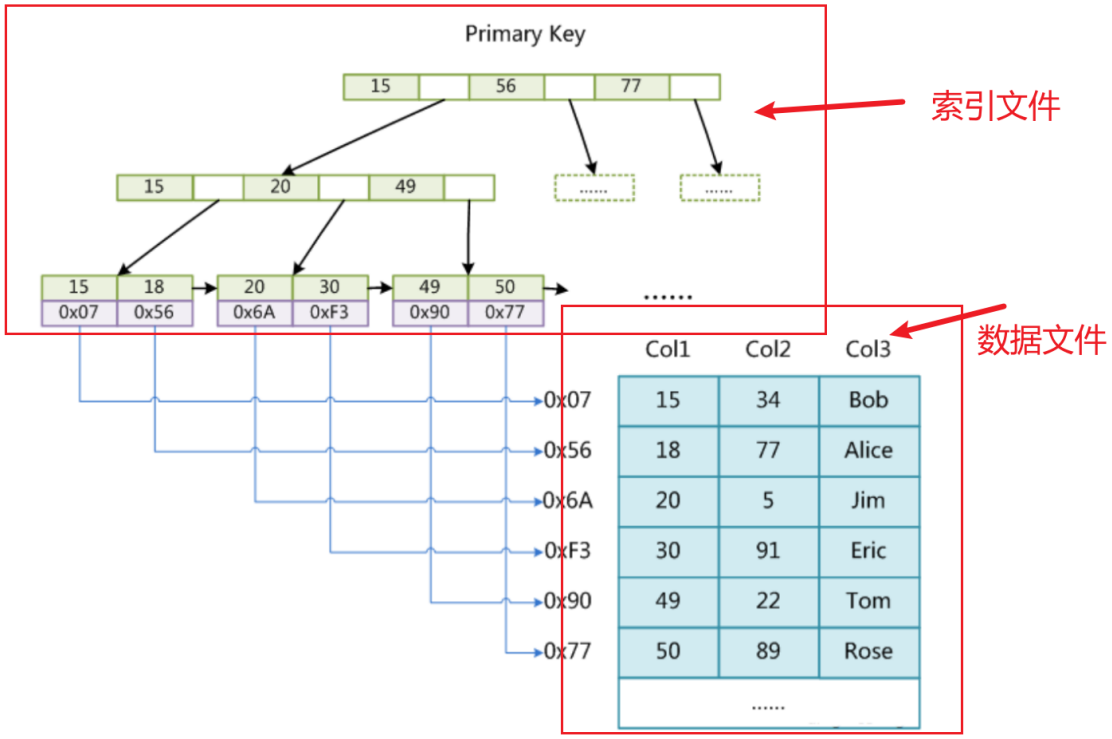
以上，我们可以看出，MySQL中数据的存储是和存储引擎息息相关的。InnoDB和MyISAM是MySQL中最常见的存储引擎。

3.2 MyISAM

这个是MySQL早期官方推出的存储引擎。是在MySQL5.1 之前默认的存储引擎。

> 主键是默认的索引列。

主键索引



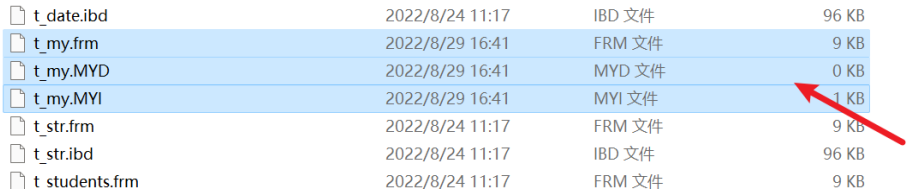
key：主键值

data：主键这一行数据对应的地址值

> MyISAM的主键索引，数据和索引没有在一块，是非聚集索引

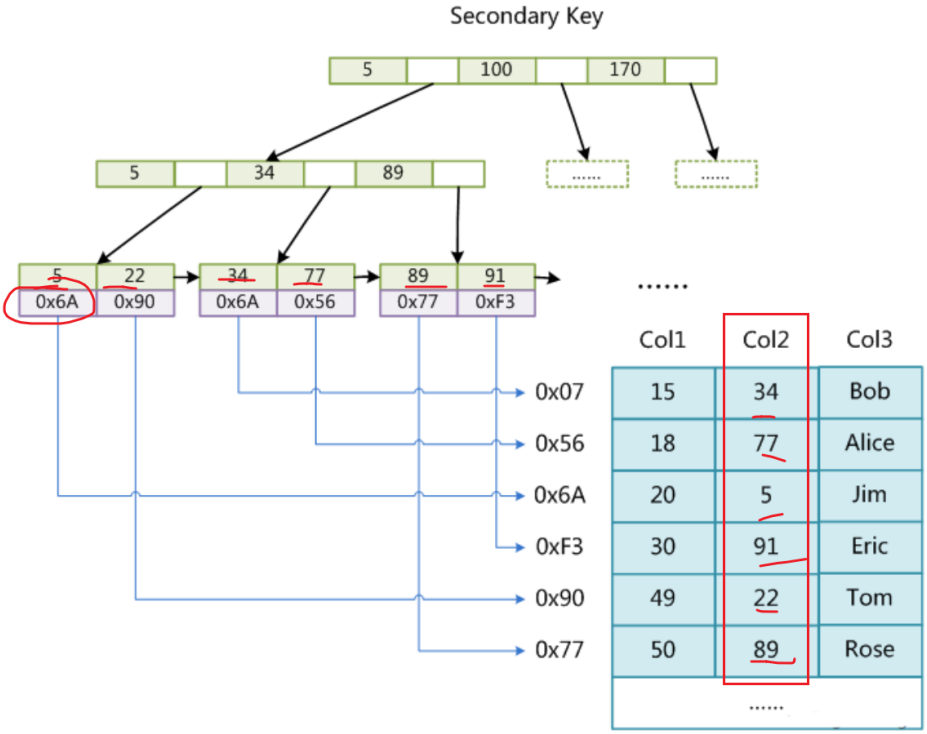
MySQL数据的存储路径：`C:\ProgramData\MySQL\MySQL Server 5.7\data`

> MyISAM存储引擎对应的表有三个文件：
>
> - .frm：表结构定义文件
> - .MYD：数据文件
> - .MYI：索引文件（这个文件中可能有多个B+树）



非主键索引

什么叫非主键索引呢？其实就是根据除了主键以外的其他的字段来建立索引树（B+树）。

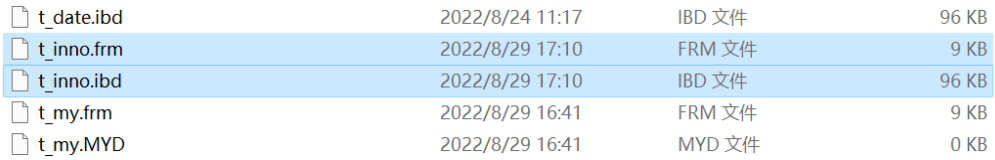


- key：索引列的值
- data：地址值

> MyISAM的非主键索引是非聚集索引。

3.3 InnoDB

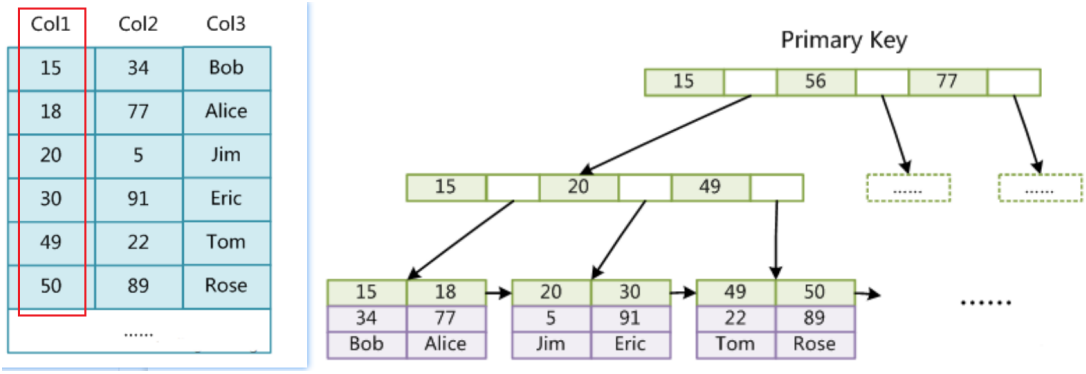
InnoBD在最早期是以插件的形式存在在MySQL中的，后来到了MySQL5.5之后，被官方设置为默认的存储引擎。



- .frm：表结构定义文件

- .ibd：数据和索引文件

主键索引

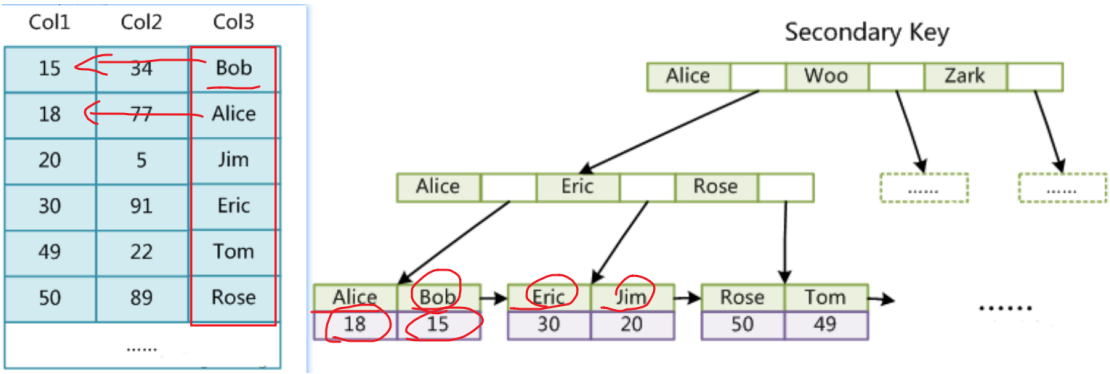


key：主键值

data：主键一行数据对应的其他的列

> 数据和索引是存储在一起的，是连续存储的，是聚集索引。

非主键索引



key：索引列的值

data：主键值

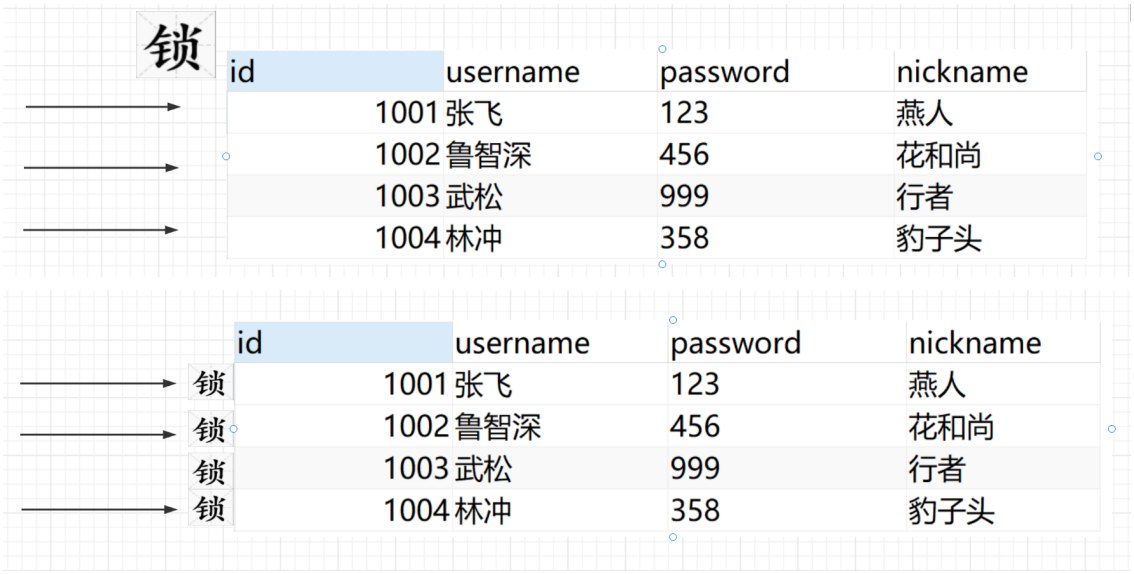
> 由于只有部分数据和索引是存储在一起的，这种也是非聚集索引。

在InnoDB中，**表中的所有的数据是依赖于主键索引树来存储的。每一个表都必须有一个主键索引树**；再换句话来说，其实就是每一个表必须得有主键。

3.4 InnoDB和MyISAM的区别

- **InnoDB支持事务，MyISAM不支持事务。其实在InnoDB中，会把每一条SQL语句自动封装成一个事务，自动提交，影响速度。**
- InnoDB支持外键，MyISAM不支持外键
- InnoDB对应的表有两个文件(.frm | .ibd)，MyISAM对应的表有三个文件(.frm | .MYI | MYD)
- InnoDB支持行锁和表锁，MyISAM只支持表锁

  (锁的粒度越细，效率越高)



表锁：锁的对象是一整张表的数据

行锁：锁的对象是一行数据

对于操作这张表来说，表锁的效率是明显没有行锁的效率高的。

那么如何去选择InnoDB和MyISAM呢？其实就是看这个表有没有事务的需求。就是说这个表示只有查询的需求还是有增删改查的需求；因为增删改会涉及到事务，所以假如一个表只有查询的需求的话，那么就没有事务的需要，这个时候就可以考虑使用MyISAM。什么样的表只有查询的需求呢？假如一个表存储的数据是不会变的，那么可以考虑使用MyISAM。什么样的数据是不变的呢？历史数据。

4. 语法

> 主键列 是天然的索引列。主键列不需要额外声明是索引，默认跟根据主键这一列建立一个B+树，叫做主键索引树。

```
# 查询表的索引
show index from tableName;

# 创建索引列  创建索引是在创建b+树，当然也会耗时！！！
create table t_user(
    id int PRIMARY KEY,
    name varchar(20),
    nickname varchar(20),
    age int,
    -- 声明name这一列是索引
    index index_name(name) using btree
)ENGINE=InnoDB character set utf8;

--

-- 删除索引
alter table t_user drop index index_name;

-- 增加索引
alter table t_user add index idx_nickname(nickname);
```

5. 面试问题

1. 索引采用的是什么数据结构？为什么采用这种数据结构

> B+树。

2. 数据库为什么推荐自定义主键，并且在MySQL中使用推荐使用主键自增的策略？

> 1. 默认的存储引擎是InnoDB
> 2. 在InnoDB中，数据是依托于主键索引树来存储的，所以对于InnoDB的表来说，必须得有主键
> 3. 推荐用户自己定义主键是因为假如用户使用了InnoDB自己维护的隐藏列来当做主键的话，那么就相当于浪费了主键索引树的查询性能，查询起来不太方便
>
> 为什么推荐主键自增呢？
>
> - 因为主键自增，就会导致B+树永远只有右边的结构发生改变，并且改变的幅度是比较可控的，利于插入效率的稳定（提高插入的效率）

3. InnoDB和MyISAM有什么区别？什么情况下使用MyISAM？
4. 什么是回表？如何避免回表？

> 假如一个查询，需要先根据非主键索引查询主键的值，再根据查询出来的主键的值查询主键索引树，这个过程就叫做回表（查询了两遍索引树）。

> 如何避免回表？（提高效率）
>
> - 尽量避免写`select *`
> - 优先考虑通过主键来查询
> - 可以考虑建立联合索引
>
>> 联合索引其实就是根据两列或者是多列的值共同建立一个索引树

5. 索引性能这么好，是不是一个表建立的索引越多越好？

> 不是。

> 每一个索引都对应一个B+树。假如表的索引变多了之后，那么就意味着B+树变多，需要维护数据的地方就变多了，同一个数据可能在多个索引树中都需要维护，这个时候维护成本会升高。

> 单个表的索引数量不要超过5个。（每个公司有自己的规定）

6. 什么样的列适合建索引呢？
    1. 查询需求比较频繁的（where后面的条件）
    2. 连接查询的条件
    3. 不重复的