* 索引 B-Tree
  1. 索引：查询条件—函数—>地址
  2. B-Tree：点查询、范围查询
  3. 根节点->中间节点->叶子节点
* B-Tree索引的特性
  1. 阶（节点大小）：一个节点能包括的最大键值数n
  2. 任何一个节点阶的个数>n/2向上取整-1：保证平衡性
* 索引的创建与使用
  1. 创建索引: db.myCollection.createIndex({‘name’: 1})
  2. 创建索引的原则：
     + 常用属性
     + 属性稳定（不常被修改）
     + 索引有效性（例如姓名是好的索引、性别是不好的索引）
  3. 使用索引可以避免使用Inode结构（主索引）
* B树相比于平衡二叉树的优势：树更低，IO代价更低（内存和硬盘间）-

**第*1*题：以下哪个因素不会显著影响B树的访问性能？**

A：树的高度

B：树的阶

C：节点的空间大小（通常一个节点为存储空间中的一页，因此可理解为页的大小）

D：节点内部的数据充满度√

B树访问性能主要由IO性能决定，ABC都能对IO性能产生影响（IO：输入和输出）

**第*2*题：B树的平衡性主要由哪条性质保证？**

A：每个节点的大小固定

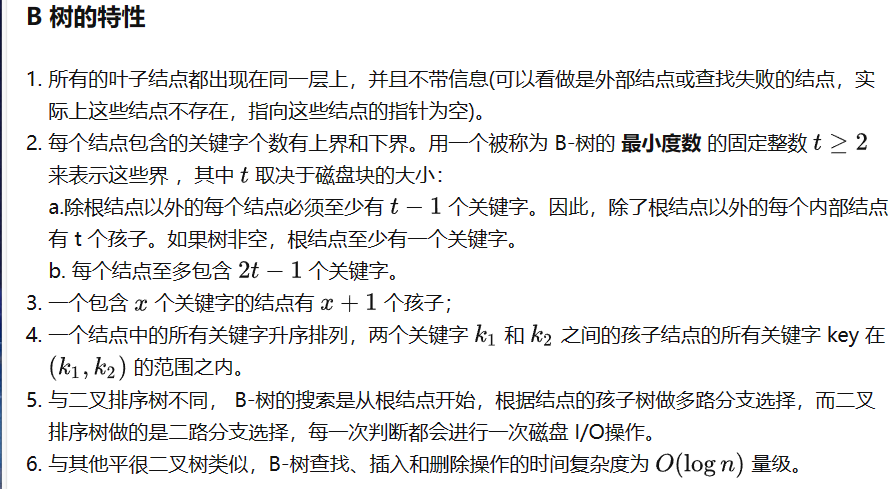
B：每个节点的充满度都超过1/2√

该性质保证树的平衡性

C：叶子节点上的数据是有序的

D：以上性质都不能

数据库使用的结构：B-Tree、Hash（、R-Tree、LSM\_Tree）



**第*3*题：如果我们在属性price上创建一个索引（比如使用指令db.myColl.createIndex( { price: 1 }) ），那么以下哪个查询可以无法从这个索引获益？**

A：db.myColl.findone({ category:"apple", price:20 })

B：db.myColl.findone({ category:"apple" })√

C：db.myColl.findone({ price:{$gte:20, $lte:30} })

D：db.myColl.findone({ category:"apple", price:{$gte:20, $lte:30} })

**第*4*题：如果我在多个属性上创建一个复合索引，例如db.myColl.createIndex({ score: 1, price: 1, category: 1 })，那么以下哪个查询无法从索引获益？**

键值：score的字节数+price的字节数+category的字节数

假设score的大小为4字节，price的大小为4字节，category的大小为2字节，此时键值为10

A：db.myColl.find({ category:"apple", price:20, score:5 })

包含score：5，可以定位区间

和其他选项对比，比单属性索引效果好

B：db.myColl.find({ score:{$gte:4} })  
包含score:{$gte:4}，可以缩小区间

C：db.myColl.find({ category:"apple", price:{$gte:20, $lte:30} })√

复合索引是有序的，不对score进行查询时，查询的结果仍然散落在整个区间内，查询无法从索引获益

D：db.myColl.find({ category:"apple", score:{$gte:4} })

包含score:{$gte:4}，可以缩小区间

**第*5*题：请问以下哪种情况最适合使用索引？**

A：属性a常用作查询条件，属性b频繁被修改。在a上创建索引。√

B：属性a常用作查询条件，属性b频繁被修改。在b上创建索引。

C：属性a常用作查询条件，文档频繁被插入和删除。在a上创建索引。

D：属性a常用作查询条件，属性a频繁被修改。在a上创建索引。

好处和代价兼有