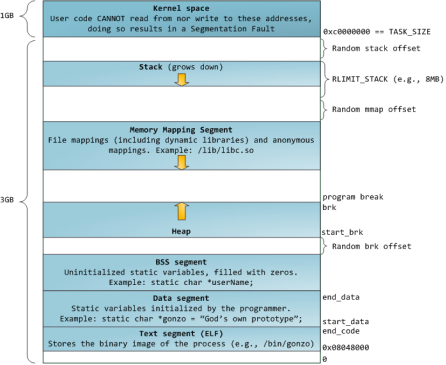
**数据结构和数据库**



堆和栈区别

堆是从低🡪高生长

栈是从高🡪低生长

取数是从低地址获取

# 优先队列怎么实现的

最大堆：根节点的值是最大值

通过数组形式存储（没有指针占用内存比是普通树小）；优先队列堆中不需要整棵树是有序的。所以不需要通过左旋右旋等操作使得平衡。

优先队列在当层所有接单被填满之前不允许开下一层。

如何维护一个优先堆

Shftup（）：如果子节点大于父节点；交换父子位置

Shiftdown(): 如果子节点小于父节点；交换父子位置

增加数据，按顺序加在最后；如果比父节点大，就递归 调用shiftup

删除根节点，把最后一个移上来，递归适用shiftdown

# 二叉搜索树BST树

小数在左节点，大数在右节点

删除节点怎么实现？

如果待删除节点是0度，直接删掉

如果是1度和二度  
1.左子树中最大值，修改删除值为左max

2.删除左max节点(递归实现）

最差查询复杂度O(n)

# AVL树

1. 左右节点高度差不超过1
2. 左边高度大了就右旋

画个图就清楚了

BinaryTree\* L=p->lchild; //左节点变成根

p->lchild=L->rchild; //左节点的右节点变成根的左节点

L->left=p; //根节点变成左节点的右节点

1. 右边高度大了就左旋

BinaryTree\* R=p->rchild; //右节点变成根

p->rchild=R->lchild; //右节点的左节点变成根的右节点

L->left=p; //根节点变成左节点的右节点

# √红黑树

1. 节点非黑即红
2. 根节点和NULL节点是黑的
3. 红色节点的子节点必须是黑，黑色节点的子节点可黑可红
4. 任何一个节点到NULL节点的路径包含黑色节点的个数相同

也是通过左旋，右旋达到平衡的

不是完全的平衡，左右树高度差不会超过两倍

Avl树适合查找为主的业务，增删回进行多次调整，影响到时间复杂度

红黑树牺牲了严格的平衡，可应用于增删查找业务，只是查找的效率没有avl高

红黑树的应用有map和set

数据结构



将一个节点插入到红黑树中，需要执行哪些步骤呢？首先，将红黑树当作一颗二叉查找树，将节点插入；然后，将节点着色为红色；最后，通过旋转和重新着色等方法来修正该树，使之重新成为一颗红黑树。

# 红黑树相对于哈希表，在选择使用的时候有什么依据？

查找速度，数据量，内存使用，可拓展性，操作

1. 哈希表查找速度为O（1）（hash还有hash函数耗时），红黑树是O(lgn).

数据量非常大的时候，hash查询效率高于红黑树

1. 对内存有严格控制的时候。hash对象特别多就会消耗大量内存（而且hash构造速度慢）
2. 如果数据是静态的（插入删除较少）用hash
3. 实际系统中，比如需要使用动态规则的防护墙系统。使用红黑树而不是散列表被证明有更好的伸缩性。Linux内核在管理vm\_area\_struct时就是采用红黑树维护内存块的。

当向 Hashtable 添加元素时，Hashtable 的实际加载因子将增加。当实际加载因子达到指定的加载因子时，Hashtable 中存储桶的数目自动增加到大于当前 Hashtable 存储桶数两倍的最小质数。

# 什么是数据库索引？数据库索引有哪些？

就跟字典目录一样。没有目录区缩小查询范围，就要整个地址遍历。

起到管理数据作用，加快查找速度作用

大类有B+树索引和hash索引

Hash索引和B+tree索引的区别。

1. 在查询速度上，只要不存在冲突。HashO(1); 如果键值不是唯一的就会遍历链表。Hash索引不会用在重复值多的列上（比如性别）
2. Hash索引是无序的，如果是范围查询索引。及时是有序的键值，经过hash算法后，也会变成不连续的。B+树叶子节点形成有序链表便于范围查询。
3. Hash无法做模糊查询。B+树索引具有最左前缀匹配。可以进行部分模糊匹配
4. B+树叶子节点形成的链表是有序的关键字。可以用于排序。Hash不具有排序
5. 存在hash冲突，在大量重复键的情况下。Hash索引的效率低。B+树查询性能稳定。

# 为什么有B树/B+树的诞生？？？用于数据库索引

https://blog.csdn.net/qq\_35571554/article/details/82796278

 数据库文件存储都是以磁盘文件存储在系统中的；

从磁盘文件中读取数据是比较耗时的，数据库的select操作的时间，取决于执行磁盘IO的次数，因此尽量减少磁盘IO就可以显著的提升数据的查询速度；

**当访问一个地址数据的时候，与其相邻的数据很快也会被访问到。每次磁盘IO读取的数据我们称之为一页（page）4k~8k；**

所以节点数越少IO读写操作少，查询速度越快

数据库索引为什么选择B树和B+

链表O(n)

数组O（1）查询，但是内存有限放不下大索引

平衡树二叉树，（lgn）但是读写磁盘次数多，查询慢。

综上，使用了B树和B+

如果一个节点由100个建，三层就有100万个数据。如果用二叉树需要20层。读写效率相差非常大

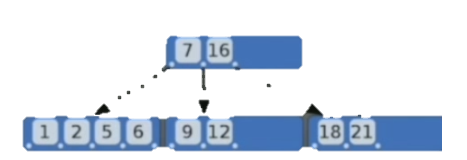
# B树

https://www.bilibili.com/video/BV1Aa4y1j7a4?t=1488

1. 阶数：每个节点最多有m个子树；非根非叶节点至少有m/2个分支

5阶B树：每个节点最多有4(m-1)个key，至少有(m+1)/2-1=2个（m/2向上取整）

阶数是人为规定的。不是单纯认为是最大节点分支数就行



1. 有n个分支的节点有n-1个关键字。
2. 按照递增排序。

查找

根节点出发，找到第一个大于等于，等于即找到

否则到合适的分支进行查找

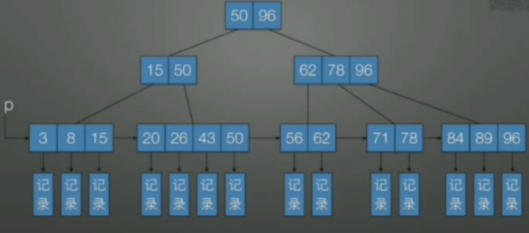
添加元素

删除关键字

节点过多，拆分节点，节点并入上一层左边或者右边

# B+树

1. N个关键字对应n个分支
2. 每个节点至少两个关键字
3. 非根节点，关键数个数（m+1）/2~m-1
4. 叶子界定啊包含信息，包含了全部关键字，叶子节点引出的指针指向记录



# B+和B不同地方

1. B+树最终所有的非叶子节点仅仅起到一个索引作用（包含对应子树的最大关键字和指向子树的指针），不包含关键字对应记录的存储地址；（索引信息）

而在B树中每个关键字对应一个记录的存储地址。（内容信息）

1. B+树上有个指针p指向关键字最小的叶子系欸但。所有叶子节点连接成一个线性链表。

因此B+树可以根据key进行顺序查找，但是b树（要中序遍历查找）

# B树和B+树适用场景

B树主要用于文件系统和文档型数据库mongodb

B+树主要用于mysql数据库索引

# 为什么Mysql使用B+不使用B?也不采用红黑树呢？

1. 在磁盘块大小一定的情况下，B树的关键字会存储相应内容信息，阶数没有B+树大

B+树能装下更多key。数据分散装载其他位置，由叶子节点保存存储位置。

1. 对范围查询。B+树支持链表顺序索引；B树需要中序遍历，操作麻烦
2. B树由于数据存在于节点，查询效率不稳定。B+树数据存储于叶子节点，查询稳定。

红黑树需要通过左旋右旋达到平衡，耗费时间。

红黑树牺牲了严格的平衡，降低了对旋转的要求，任何不平衡都会在三次旋转中解决。

适合用于内存中。但是内存大小有限检索范围有限，在内存中适用效率高。

数据量大，放在磁盘；对磁盘io读写次数大，查询效率低。

# 一致性哈希算法

解决是数据库物理节点增删时候数据全部发送迁移的问题

有n个数据库，对数据进行哈希运算，然后对n取模确定存放位置。

数据库新增为n+1个时，所有数据哈希运算后，对n+1重新取模

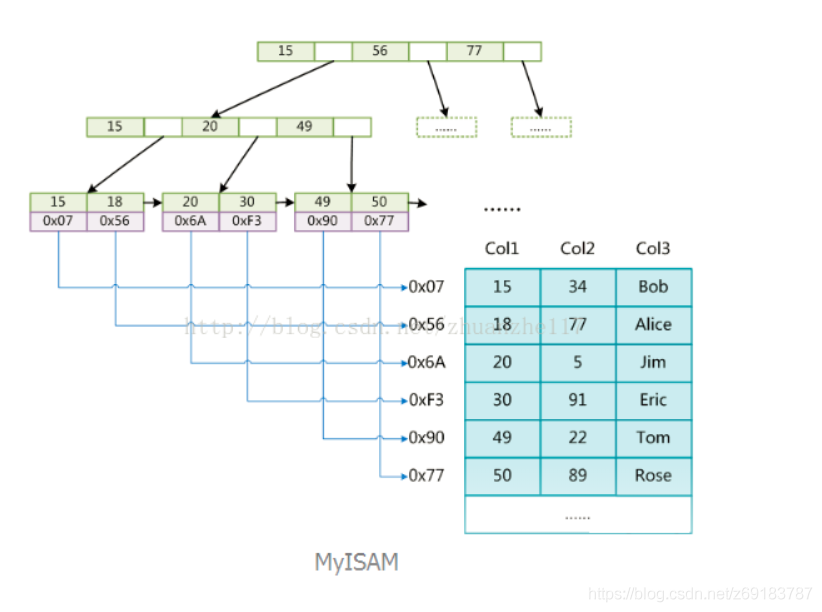
一致性哈希就是把0-2^32，0和2^32首尾闭合。每个数据分布在其中一个节点。

这样在节点上新增物理数据库，只需要部分数据迁移

# MySQL最常用的两个存储引擎

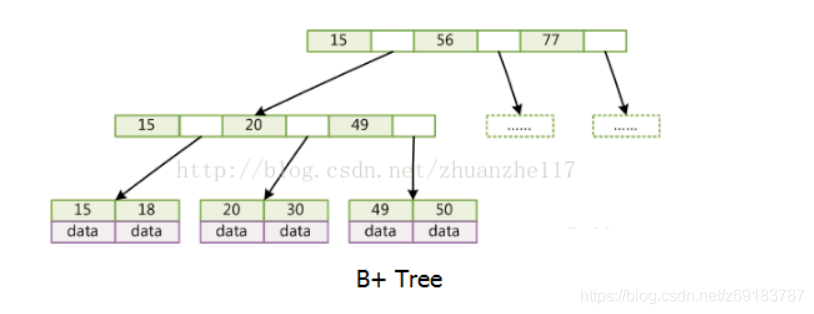
## MyISAM（Indexed Sequential Access Method）

非聚集索引



## InnoDB

聚集索引



# 两种引擎区别

索引结构上：非聚簇型/聚簇型（数据和索引文件是否放一起）；叶子节点是否构成链表

事务型，合并事务，性能高

ACID事务安全，要么全部成功，要么回滚

锁粒度级别

全文索引

查询效率（myisam相对简单，效率高）

外键支持型

* 1).**［事务］**MyISAM是非事务安全型的，而InnoDB是事务安全型的，默认开启自动提交，宜合并事务，一同提交，减小数据库多次提交导致的开销，大大提高性能。并发性强
* 2).**［锁］**MyISAM锁的粒度是表级，而InnoDB支持行级锁定。
* 3).**［全文索引］**MyISAM支持全文类型索引，而InnoDB不支持全文索引。
* 4).**［查询效率］**MyISAM相对简单，所以在效率上要优于InnoDB，小型应用可以考虑使用MyISAM。
* 5).［外健］MyISAM不支持外健，InnoDB支持。
* 6).［count］MyISAM保有表的总行数，InnoDB只能遍历。
* 6).MyISAM表是保存成文件的形式，在跨平台的数据转移中使用MyISAM存储会省去不少的麻烦。
* 7).InnoDB表比MyISAM表更安全，可以在保证数据不会丢失的情况下，切换非事务表到事务表（alter table tablename type=innodb）。
* 8）MyIsam索引和数据分离，InnoDB在一起，MyIsam天生非聚簇索引，最多有一个unique的性质，InnoDB的数据文件本身就是主键索引文件，这样的索引被称为“聚簇索引”

两者引擎应用场景

* 1).MyISAM管理**非事务表**。它提供**高速存储和检索**，以及**全文搜索能力**。如果应用中需要执行**大量的SELECT查询**，那么MyISAM是更好的选择。
* 2).InnoDB用于事务处理应用程序，具有众多特性，包括**ACID事务支持**。如果应用中需要**执行大量的INSERT或UPDATE操作，则应该使用InnoDB**，这样可以提高多用户**并发操作的性能**。

# 最左索引

a.b.c三个分别建立索引，就是普通索引或单个索引。

（a.b.c）整体建立索引，就是复合索引。因为“最左前缀原则”所以其实相当于创建了(a,b,c)，(a,b)、(c)三个索引。因此我们在创建复合索引时应该将最常用作限制条件的列放在最左边，依次递减。

select \* from test where a='11'

select \* from test where a='11' and b=1

select \* from test where a='11' and b=1 and c=2.0

以上有索引

select \* from test where b=11

select \* from test where b=1 and c=2.0

以上无索引

# 事务的隔离级别

事务的四大特性ACID原子性。

A：原子性。事务是最小的业务逻辑单元

B: 一致性。一个事务必须保证多条DML语句同时成功或失败

C: 隔离性。两个事务之间具有隔离性（隔离级别问题）

D: 持久性。最终数据必须持久化到硬盘中，一个事务才算完成。

数据库中事务的四种隔离级别

* 1. Read\_uncommitted 读未提交。可读取未提交事务的操作数据，最低的隔离级别，一般都没有用的。这种情况会出现脏读。
  2. Read\_committed 读已提交。一个事务等另一个事务提交之后才可进行读取，解决了脏读问题，但会出现不可重复读。
  3. Repeated\_read 可重复读。读取事务开启的时候不能对数据进行修改，解决了不可重复读问题，但是存在幻读问题。
  4. Serializable 序列化。是最高的事务隔离级别，可以避免脏读、不可重复读与幻读。但是这种事务隔离级别效率低下，比较耗数据库性能，一般不使用；

如何合理选择隔离级别

可以根据需求设置数据库的事务的级别。“读未提交”一般没用，“读已提交”解决脏读但存在不可重复读，“可重复读”解决了脏读和不可重复读，但会出现幻读。串行化读，都可以解决，但是需要注意的是事务级别越高性能越低。

MySQL默认第三隔离级别。

Oracle默认第二隔离级别。

**1、脏读**

出现原因：一个事务读取到了缓存中另一个事务未提交的脏数据。

说明：当事务B对data进行了修改但是未提交事务，此时事务A对data进行读取，并使用事务B修改的数据做业务处理。

**2、幻读**

出现原因：一个事务在读取数据时，另一个事务插入了数据，导致上个事务第二次读取数据时，数据不一致。

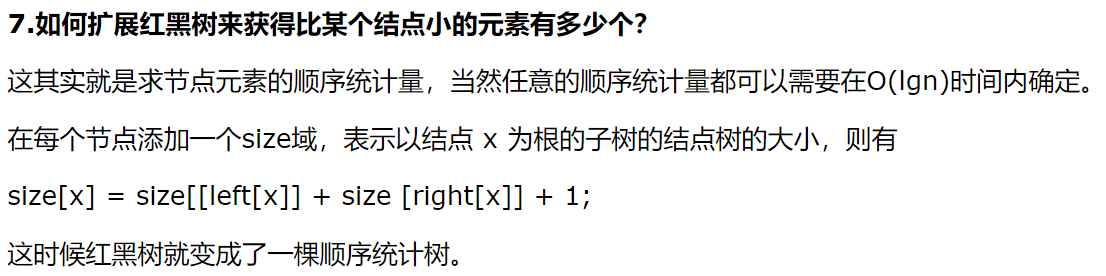
说明：data 表有一条数据，事务A对data进行读取， 事务B对data进行数据新增 ，此时事务A读取只有一条数据，而最后实际data是有两条数据，就好象发生了幻觉一样情况成为幻读；

**3、不可重复读**

出现原因：读取数据的同时可以进行修改;

说明：事务A、事务B同时对data进行访问，事务A对data进行读取，事务B对data进行修改，当事务A第一次对data进行读取完后事务B提交，此时当事务A第二次读取该数据时的数据就与第一次读取的数据不同，这种情况称为不可重复读；

# 数据场景题





<https://www.cnblogs.com/simonote/articles/3087185.html>

2. （**出现频率最高**）1亿个int的大文件，取频次最高的数，相同则取数字最大的

10^8次个int=4\*10^8byte

4G=4\*10^9 内存可以装得下

Hash map统计出现次数

要是内存放不下，就进行成n个文件，x%n分类放置

相同的数字一定会被分到相同文件

用hash表统计每个文件中的最大

1. （**频率**最高）1000亿个string查找出现频率最高

内存放不下；

对string进行编码，编码成一个数字。就变成问题1了。

编码冲突怎么解决，一共就26个字母。

如果还包含其他乱七八糟的字符怎么半。

编码分类，然后针对string 使用hash找最大

1. （**找相同**）共有两个文件，每个文件里面有1000亿个url，如何找出两个文件中相同的url？

采取分而治之的方法。   
遍历文件a，对每个url求取hash(url)%1000，然后根据所得值将url分别存储到1000个小文件（设为a0,a1,...a999）当中。这样每个小文件的大小约为300M。遍历文件b，采取和a相同的方法将url分别存储到1000个小文件(b0,b1....b999)中。这样处理后，所有可能相同的url都在对应的小文件(a0 vs b0, a1 vs b1....a999 vs b999)当中，不对应的小文件（比如a0 vs b99）不可能有相同的url。然后我们只要求出1000对小文件中相同的url即可。   
比如对于a0 vs b0，我们可以遍历a0，将其中的url存储到hash\_map当中。然后遍历b0，如果url在hash\_map中，则说明此url在a和b中同时存在，保存到文件中即可。   
如果分成的小文件不均匀，导致有些小文件太大（比如大于2G），可以考虑将这些太大的小文件再按类似的方法分成小小文件即可。

1. （**频率**最高）网站日志中记录了用户的IP，找出访问次数最多的IP

数据太大，分治。

Hash（IP）%1024分类成小文件

相同ip肯定分到同一个文件

然后使用hash统计频率，

1024个数字比较然后取最大，对应的那个IP

1. （**topK大小**）N个数求topK小的数

一亿个浮点数，找最大的一万个？

排序的话，O(nlgn)

只关心前k个。利用优先队列。

如果是找最大前K个，用最小堆；最小前K个用最大堆

O(nlogk)

# 数据库

**表示时间值和日期和时间类型为：DATETIME\ DATE\ TIMESTAMP\ TIME\ YEAR**

**字符串类型：CHAR\ VARCHAR \BINARY \VARBINARY \BLOB \TEXT \ENUM\ SET**

**主键**：唯一，不适用任何业务相关（身份证，手机号）的字段；一般是id/guid

Guid（一种算法生成一段唯一的数）

自增整数类型：最多保存21亿数据

BIGINT：定义主键为BIGINT NOT NULL AUTO\_INCREAMENT类型

**联合主键**

关系数据库实际上还允许通过多个字段唯一标识记录，即两个或更多的字段都设置为主键。

在没有必要的情况下，尽量不要使用联合主键，因为它给关系表嗲来了复杂度的上升。

**外键**

在student表中，通过class\_id的字段，可以把数据与另一张表关联起来。这种列称做外键。

关系数据库通过外键可以实现一对多、多对多和一对一的关系。

可以通过数据库约束，也可以不设置约束，仅依靠应用程序的逻辑保证

**最左索引**（hash不行，B+/B可以）



Alter table students

**Add index** idx\_score(sore) 增加了关于scores字段的索引 idx\_score

索引的效率取决于索引列的值是否为散列表，该列的值如果越互不相同，索引效率越高。

对性别创建索引就没有意义。

使用主键索引可以保证唯一性，效率高

唯一索引

唯一的列（身份证、邮箱地址等），由于具有业务含义，因此不宜作为主键。但是，这些列根据业务要求，又具有唯一性约束（两个记录不能出现同一个身份证）

**ADD UNIQUE INDEX** uni\_name(name) 为该列添加唯一索引

只是添加唯一约束，不创建唯一索引

ALTER TABLE students

**ADD CONSTRAINT** uni\_name UNIQUE(name)

小结：

通过数据库表创建索引，可以提高查询速度。

通过创建唯一索引，可以保证某一列的值具有唯一性。

数据库索引对用户和程序来说都是透明的

索引需要维护和占用额外的物理内存，增删改查都要动态维护索引

在哪些列上创建索引？

* 1. 经常需要搜索的列上
  2. 作为主键
  3. 经常用在连接的列上
  4. 经常需要根据范围进行搜索
  5. Where子句
  6. 经常需要排序的列
  7. 外键连接的列上

不该在哪些地方创建索引

1. 查询中很少用到的列
2. 重复性高的（如性别，散列表）
3. 数据量大的（图片文本）
4. 修改性能要求远远大于搜索性能要求（反之如果修改需要同时修改索引）

按照物理存储角度对索引分为：（索引顺序和表顺序，查询效率和修改效率）

聚集索引：表记录的排序顺序和索引顺序一直，查询效率块。找到第一个索引就能获得其余连续性记录内容。缺点是修改慢，因为为了使表记录和索引的排列顺序一直，在插入记录的时候，回对数据页重新排序

非聚集索引：

表记录和索引的排列顺序不一定一致，两种索引都采用B+树的结构 ，非聚类索引的叶子层并不和实际数据页重叠，而采用叶子层包含的一个指向表记录的指针。非聚类索引层次多，不会造成数据重排。相对而言修改块。

## 数据量基本概念

## 事务、原子a，一致性c，隔离性i，持久性d

事务：数据库单独的执行单元

更新数据成功，事务中更改的数据就会提及哦。否则事务取消或者回滚，更改无效。

事务必须满足ACID四个属性

原子性

# Redis内存数据库

单reactor，单线程

由于业务简单

Nosql非关系型数据库，不用sql做检索直接使用key\_value.支持5大数据类型（集合，顺序集合，字符串，散列表，链表）

在内存中，用于缓存。加快查询速度

RedisVM机制

1. 内存不够的时候怎么处理：有自己的swap函数将冷数据调出内存。不调用OS的swap。不会线程阻塞
2. 能够保证OS不进行页切换，redis对象单位小，一个页4k可以放很多个redis对象。

List放在不同的页中，保证了每个页活页，避免OS页交换。

1. OS发生页切换，redis可以设置让工作线程完成，主线程仍然可以处理client请求
2. Redis交换到磁盘的内容进行压缩，比OS的页交换内容小很多。减少了IO操作，增大了效率。

应用场景：（存储频繁操作的信息）

缓存

计数器（计数量频繁读写

分布锁

在分布式场景下，无法使用单机环境下的锁对多个节点上的进程进行同步。可以使用redis自带的setnx命令实现分布式锁。此外可以使用官方提供的redLock分布式锁实现

为什么要用redis？

什么是redis持久化

Redis key的过期事件和永久有效怎么设置

Redis是单线程，如何提高多核cpu的利用率

直接用mysql可以吗？

Redis进阶

Key操作命令

事务

主从复制

持久化

Redis5中数据结构

Redis安装配置以及java访问redis

Memcache缓存库