**海量数据处理算法：**

**Bloom filter**:用位数组简洁的表示一个集合，并能判断一个元素是否属于这个集合，实现是用k个哈希函数将元素映射到一个m长度的数组。是一个判断元素是否属于集合的算法，可能会误判，但不会漏判

（时间空间互换之外，引入了错误率的概念，以错误率换取时空效率）

**Hash**:hash+链表

**Bit-map**:所谓的[Bit-map](http://blog.csdn.net/hguisu/article/details/7880288" \t "https://www.cnblogs.com/ECJTUACM-873284962/p/_blank)就是用一个bit位来标记某个元素对应的Value， 而Key即是该元素。由于采用了Bit为单位来存储数据，因此在存储空间方面，可以大大节省。可进行数据的快速查找，判重，删除，一般来说数据范围是int的10倍以下

0-7内的5个元素(4,7,2,5,3)排序

开辟1byte的空间，值全部标为0，将第4,7,2,5,3位标为1，顺序读出为1的位置，23456排序完成

**双层桶**：面对一堆大量的数据我们无法处理的时候，我们可以将其分成一个个小的单元，然后根据一定的策略来处理这些小单元，从而达到目的。

【问题实例】   
1).2.5亿个整数中找出不重复的整数的个数，内存空间不足以容纳这2.5亿个整数。

有点像鸽巢原理，整数个数为2^32,也就是，我们可以将这2^32个数，划分为2^8=256个区域(比如用单个文件代表一个区域)，然后将数据分离到不同的区 域，然后不同的区域在利用bitmap就可以直接解决了。也就是说只要有足够的磁盘空间，就可以很方便的解决。

2).5亿个int找它们的中位数。

这个例子比上面那个更明显。首先我们将int划分为2^16个区域，然后读取数据统计落到各个区域里的数的个数，之后我们根据统计结果就可以判断中位数落到那个区域，同时知道这个区域中的第几大数刚好是中位数。然后第二次扫描我们只统计落在这个区域中的那些数就可以了。

**倒排索引**：VSM全称是Vector Space Model(向量空间模型)，是IR(Information Retrieval信息检索)模型中的一种，由于其简单，直观，高效，所以被广泛的应用到搜索引擎的[架构](http://lib.csdn.net/base/architecture" \o "大型网站架构知识库" \t "https://www.cnblogs.com/ECJTUACM-873284962/p/_blank)中。98年的Google就是凭借这样的一个模型，开始了它的疯狂扩张之路。

文档(Document)：一个完整的信息单元，对应的搜索引擎系统里，就是指一个个的网页。

标引项(Term)：文档的基本构成单位，例如在英文中可以看做是一个单词，在中文中可以看作一个词语。

查询(Query)：一个用户的输入，一般由多个Term构成。

那么用一句话概况搜索引擎所做的事情就是：对于用户输入的Query，找到最相似的Document返回给用户。而这正是IR模型所解决的问题：

信息检索模型是指如何对查询和文档进行表示，然后对它们进行相似度计算的框架和方法。

相似算法由cos夹角法演变，并结合tf,idf

## **B-树：**

多路搜索树，每个结点存储M/2到M个关键字，非叶子结点存储指向关键

字范围的子结点；

所有关键字在整颗树中出现，且只出现一次，非叶子结点可以命中；

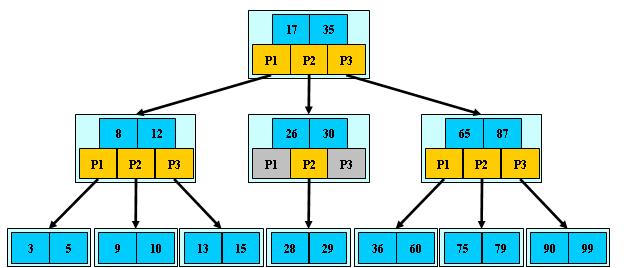
## **B+树：**

在B-树基础上，为叶子结点增加链表指针，所有关键字都在叶子结点

中出现，非叶子结点作为叶子结点的索引；B+树总是到叶子结点才命中；

## **B-树：**

## **如：（M=3）**



B-树的搜索，从根结点开始，对结点内的关键字（有序）序列进行二分查找，如果

命中则结束，否则进入查询关键字所属范围的儿子结点；重复，直到所对应的儿子指针为

空，或已经是叶子结点；

# **B-树的特性：**

1.关键字集合分布在整颗树中；

2.任何一个关键字出现且只出现在一个结点中；

3.搜索有可能在非叶子结点结束；

4.其搜索性能等价于在关键字全集内做一次二分查找；

5.自动层次控制

## **B+树是B-树的变体，也是一种多路搜索树：**

1.其定义基本与B-树同，除了：

2.非叶子结点的子树指针与关键字个数相同；

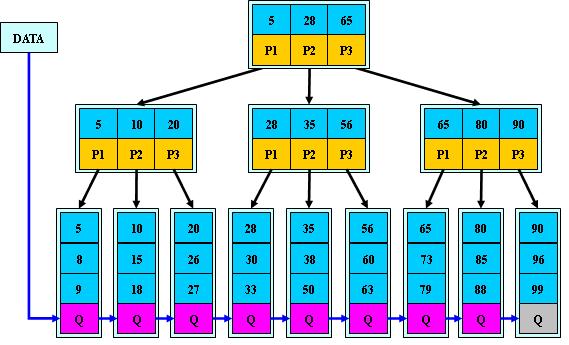
3.非叶子结点的子树指针P[i]，指向关键字值属于[K[i], K[i+1])的子树

（B-树是开区间）；

5.为所有叶子结点增加一个链指针；

6.所有关键字都在叶子结点出现；

**如：（M=3）**

  
B+的搜索与B-树也基本相同，区别是B+树只有达到叶子结点才命中（B-树可以在

非叶子结点命中），其性能也等价于在关键字全集做一次二分查找；

# **B+的特性：**

1.所有关键字都出现在叶子结点的链表中（稠密索引），且链表中的关键字恰好

是有序的；

2.不可能在非叶子结点命中；

3.非叶子结点相当于是叶子结点的索引（稀疏索引），叶子结点相当于是存储

（关键字）数据的数据层；

4.更适合文件索引系统；

**缺点：**   
通常数据量会非常大，磁盘中的数据採用这样的分页形式的话，就会比較多。非常有可能存储的数据在两个页表其中不连续，相隔非常远，这样的顺序查询的方式就会比較慢。   
B+树最大的性能问题是**会产生大量的随机IO**。随着新数据的插入，**叶子节点会慢慢分裂，逻辑上连续的叶子节点在物理上往往不连续，甚至分离的非常远**，**但做范围查询时。会产生大量读随机IO。**   
对于大量的**随机写也一样**，举一个插入key跨度非常大的样例。如7->1000->3->2000 … 新插入的数据存储在磁盘上相隔非常远，会产生大量的随机写IO。从上面能够看出，低下的磁盘寻道速度严重影响性能。

为了**让读性能尽量高，**数据在磁盘中必须得有序。这就是B+树的原理，可是写就悲剧了，由于会产生大量的随机IO，磁盘寻道速度跟不上。  
LSM Log-Structured Merge-Trees树本质上就是在读写之间取得平衡，和B+树相比，**它牺牲了部分读性能。用来大幅提高写性能。**它的原理是把一颗大树拆分成N棵小树， 它首先写入到内存中（内存没有寻道速度的问题，随机写的性能得到大幅提升），在内存中构建一颗有序小树，**随着小树越来越大，内存的小树会flush到磁盘上。**当读时，由于不知道数据在哪棵小树上。因此必须遍历全部的小树，但**在每颗小树内部数据是有序的**。

## ****1.HBase与关系数据库比较****

#### ****1）行式数据库****

优点:1.数据存储在一起。2.INSERT/UPDATE数据较容易。   
缺点:1.选择操作（select）时，即使是几行所有数据也要被读取

#### ****2）列式数据库（hbase）****

优点：1.查询时只有涉及的列才会被读取。2.投影很高效。3.任何列都会可以作为索引。   
缺点：INSERT/UPDATE比较麻烦。

**行式数据库**是按照行存储的，行式数据库擅长随机读操作不适合用于大数据。像SQL server,Oracle，mysql等传统的是属于行式数据库范畴。

**列式数据库**  同一个列的数据的格式比较类似，这样可以做大幅度的压缩。这样节省了存储空间，也节省了I/O,因为数据被压缩了，这样读的数据量随之也少了。

同一个列的数据会挤在一起，比如挤在block里，当我需要读某个列的时候，值需要把相关的文件或块读到内存中去，整个列就会被读出来，这样I/O会少很多。

数据库以行、列的二维表的形式存储数据，但是却以一维字符串的方式存储，例如以下的一个表：

|  |  |  |  |
| --- | --- | --- | --- |
| EmpId | Lastname | Firstname | Salary |
| 1 | Smith | Joe | 40000 |
| 2 | Jones | Mary | 50000 |
| 3 | Johnson | Cathy | 44000 |

这个简单的表包括员工代码(EmpId), 姓名字段(Lastname and Firstname)及工资(Salary).

这个表存储在电脑的内存(RAM)和存储(硬盘)中。虽然内存和硬盘在机制上不同，电脑的操作系统是以同样的方式存储的。数据库必须把这个二维表存储在一系列一维的“字节”中，由操作系统写到内存或硬盘中。

行式数据库把一行中的数据值串在一起存储起来，然后再存储下一行的数据，以此类推。

1,Smith,Joe,40000;2,Jones,Mary,50000;3,Johnson,Cathy,44000;

列式数据库把一列中的数据值串在一起存储起来，然后再存储下一列的数据，以此类推。

1,2,3;Smith,Jones,Johnson;Joe,Mary,Cathy;40000,50000,44000; 这是一个简化的说法。

****2 Hbase的局限：****

1、只能做简单的Key value查询，复杂的sql统计做不到。

2、只能在row key上做快速查询。

1.1、Hbase适合大量插入同时又有读的情况

1.2、 Hbase的瓶颈是硬盘传输速度，Oracle的瓶颈是硬盘寻道时间。

 Hbase本质上只有一种操作，就是插入，其更新操作是插入一个带有新的时间戳的行，而删除是插入一个带有插入标记的行。其主要操作是收集内存中一批数据，然后批量的写入硬盘，所以其写入的速度主要取决于硬盘传输的速度。Oracle则不同，因为他经常要随机读写，这样硬盘磁头需要不断的寻找数据所在，所以瓶颈在于硬盘寻道时间。

根据事实表和维度表的关系，又可将常见的模型分为**星型模型**和**雪花型模型**。  
**星形模型：**当**所有维度表连接到事实表上的时候**，整个图就像一个星星，故称之为星型模型。星型架构是一种非正规化的结构，多维数据集的每一个维度都直接与事实表相连，不存在渐变维度，所以数据有一定冗余**。因为有冗余，所以很多统计不需要做外部的关联查询**，因此一般情况下效率比雪花模型高。

**雪花模型：**当有**多个维度表没有直接连接到事实表上**，而是通过其他维度表连接到事实表上时，其图形就像雪花，故称雪花模型。**雪花模型的优点是减少了数据冗余，所以一般情况下查询需要关联其他表。**在冗余可接受的前提下使用星型模型。

**星型模型和雪花模型的区别在于：维度表是直接连接到事实表还是其他维度表。**

第三个区别在于性能的不同。**雪花模型在维度表、事实表之间的连接很多，因此性能方面会比较低。**举个例子，如果你想要知道Advertiser 的详细信息，雪花模型就会请求许多信息，比如Advertiser Name、ID以及那些广告主和客户表的地址需要连接起来，然后再与事实表连接。而星形模型的连接就少的多，在这个模型中，如果你需要上述信息，你只要将Advertiser的维度表和事实表连接即可。

雪花模型使得维度分析更加容易，比如“针对特定的广告主，有哪些客户或者公司是在线的?”星形模型用来做指标分析更适合，比如“给定的一个客户他们的收入是多少?”