## Q1

计算需要多少个block去存储关系，

解：通过题目知道每个block能存多少个字节，然后需要知道每个tuples有多少个字节，然后 ⎣每个block存储的字节/每个tuples有多少个字节⎦先得出每个block最多能存储多少条tuples. 再用⎡总tuples数 / 每个block能装几条tuples数⎤得出最终答案。

范例回答: each tuple of student is 350 bytes. Each block at most holds ⎣4096 bytes / 350 bytes⎦ = 11 tuples. There are 25000, so ⎡25000 tuples / 11 tuples per block⎤ = 2273 blocks required.

计算稠密索引的高度：logn/2(K)

计算存储主索引指针所需要的块数(sparse, one index entry for one block)

解: 先计算key属性的大小+每个key指针的大小= 总每个需要的大小。再计算每个block最多可以存储⎣ 每个block能存的大小/ 每个key需要的总大小⎦ 个 index entries. 最后计算一共需要多少个block去存储: ⎡该关系需要的block数量 / 每个block 能存储的指针数量⎤

范例回答：Each index is 10 bytes for the key plus 10 for the pointer(20 bytes in total). Each block at most can store ⎣4096 bytes / 20 bytes⎦ = 204 index entries. There are 2273 blocks in the student relation(answer of the pervious question) so, ⎡2273 / 204⎤ = 12 blocks are need.

插入或删除b+树

插入： 第一步看所插的节点有没有位置，如果有则直接插，如果没有位置，则将节点分裂，并且将分裂后右边节点的第一个key上移给父亲，如果父亲没满则直接插入，如果父亲也蛮了，则分裂该节点并且将分裂后左边节点的最后一个key提上去，并且将这个key删掉。

删除：第一步看所删除的节点，观察删除后数量是否>⎡(n-1)/2⎤如果是的话则直接删除。如果不是则考虑与相邻节点合并，如果邻居节点满了，就向邻居节点要一个过来，左右都行。然后处理下父亲节点的索引。

Hash桶：

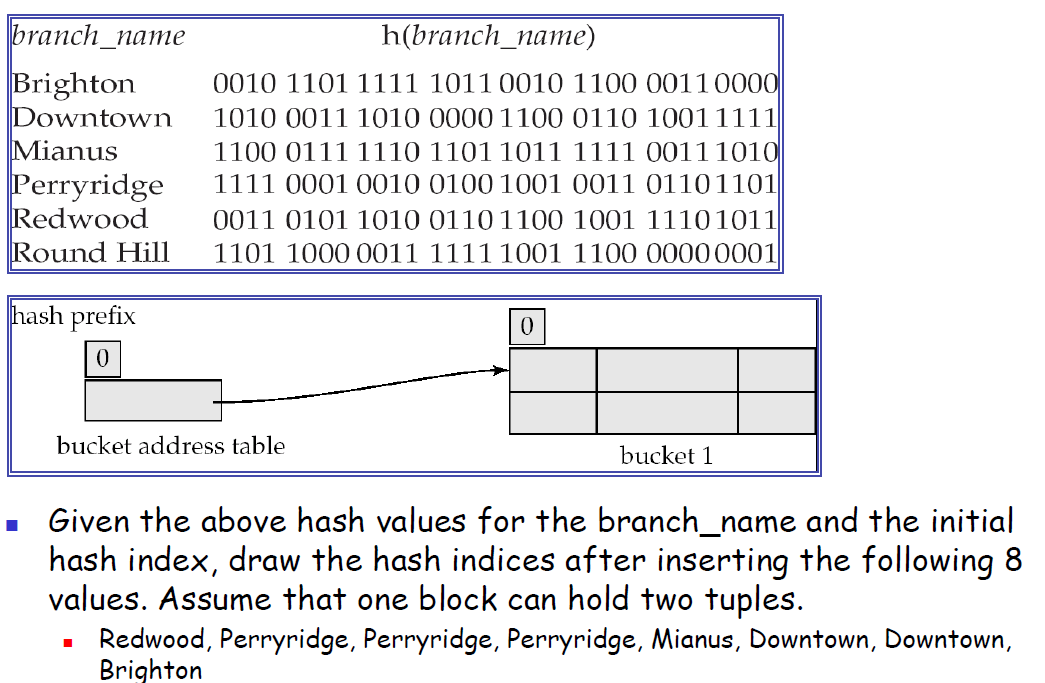
动态hash插入：第一种是把散列表都给定了，然后一个一个往里带入就行了，一个是给了哈希函数然后自己算，mod就是取模，再然后换成二进制数。往里插入。规则1：第一次插的时候不用分桶。2：如果插入的东西桶里已经有了，就在后面加一个溢出桶。3：如果插入时发现桶未满，且前几位数一样，则直接插，如果发现满了，就扩大桶，并且也扩大哈希表（要成倍数增加）。

Q1还有的问题就是删除时，只是删除叶节点的值，而非叶节点没有管。

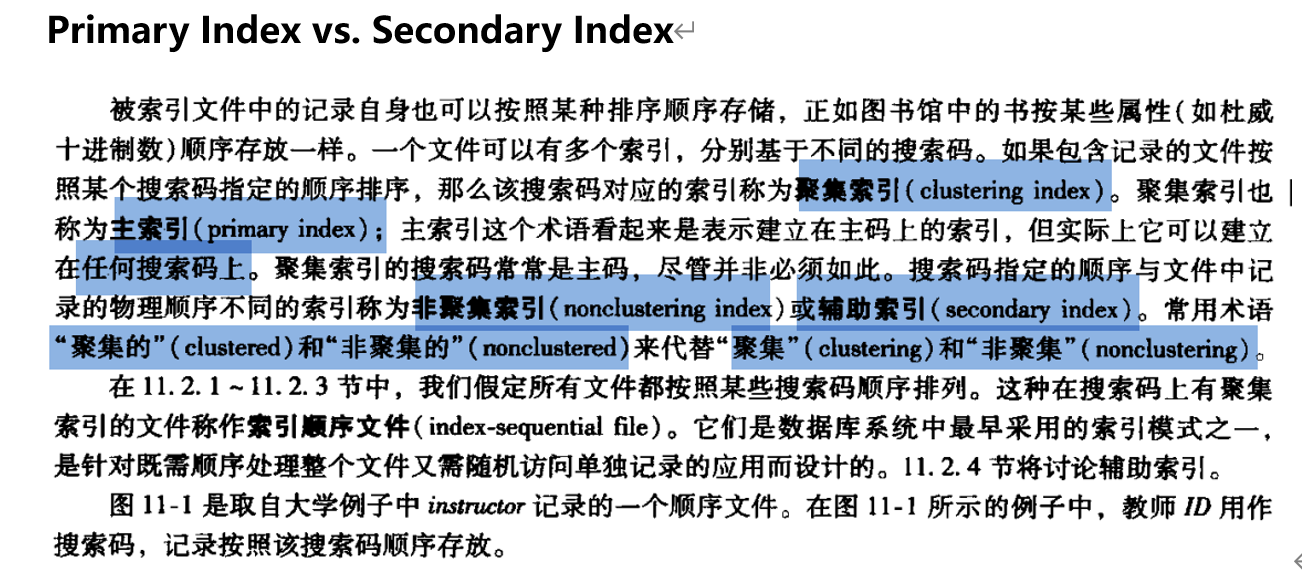
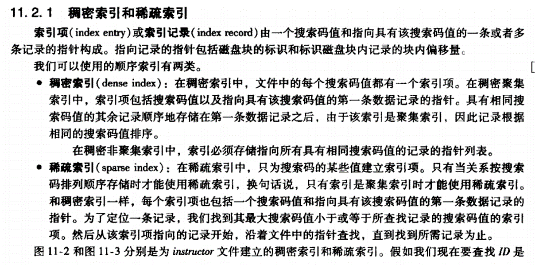
就如果直接把节点删除了了话，父节点的那个值也要删掉，如果只是删掉一个search key的话父节点不用管！

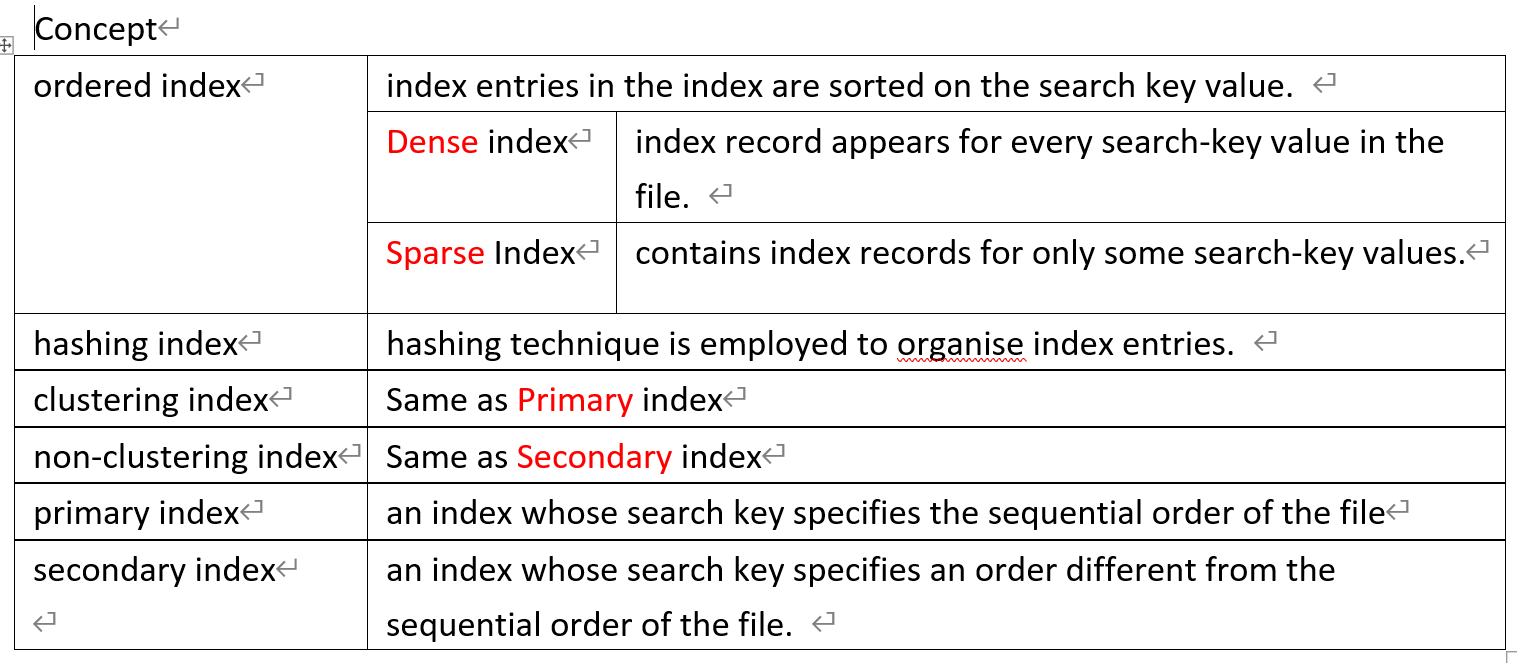
如果使用簇索引的话，怎么计算关系需要的块的大小。

桶如果说多个指针指向一个桶的话，该桶再插入的话桶分裂，那桶地址表要分裂吗



概念：





静态散列索引的优势：

One disadvantage of sequential file organization is that we must access the index structure to locate the data, or we must use a binary search, which will lead to excessive I/O operations, and the file organization of hash technology allows us to avoid accessing the index structure.

动态散列优势：

Most databases will grow larger over time.

The dynamic hashing technology allows the hash function to be changed dynamically to adapt to the increase or decrease of the database.

Q2：

我们用**传送磁盘块数**以及**搜索磁盘次数**来度量查询计划的代价。假设磁盘系统传输一个块的平均消耗tT秒，磁盘块平均访问时间（磁盘搜索时间加上旋转延迟）为tS秒，则一次传输b个块以及S次磁盘搜索将消耗b\*tT+S\*tS秒。tT和tS的值必须针对所使用的磁盘系统进行计算（意思就是给定的，不是你算出来的），通常典型数值是tT =0.1毫秒和tS=4毫秒。**我们每次选较小的块作为外层关系，也就是nr,br作为外层关系**

*嵌套连接循环*：

最坏的情况(缓冲区只能容纳每个关系的一个数据块): 我们需要**nr\*bs+br**次块传输，需要**nr+br**次磁盘搜索次数。

最好的情况(内存有足够的空间同时容纳两个关系,此时每一数据块只需要读一次): 我们需要br+bs次块传输，加上2次磁盘搜索。

*块嵌套循环连接：*

最坏的情况: 我们需要**br\*bs+br**次块传输，2br次磁盘搜索。

最好的情况：我们需要**br+bs**次块传输，2次磁盘搜索。

*索引嵌套循环连接：*（将有B+树的那一个作为内循环，则下面的公式带入的值是外循环的tuples和blocks. 如果两个都有b+树，则选取元组树较少的作为外层关系）

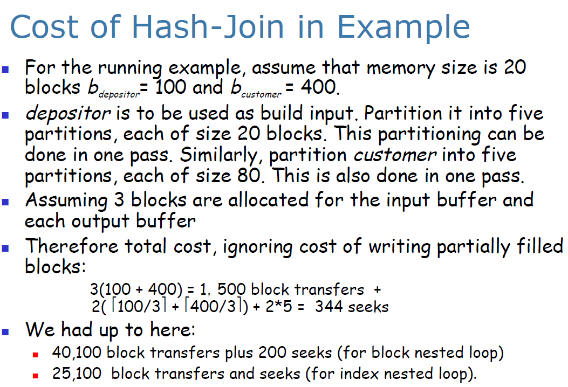
块传输次数和磁盘搜索次数一样：nr\*c+br  (c = (hi+1), 也就是树的高度加1),c是使用连接条件对关系s进行单次选择操作的代价。

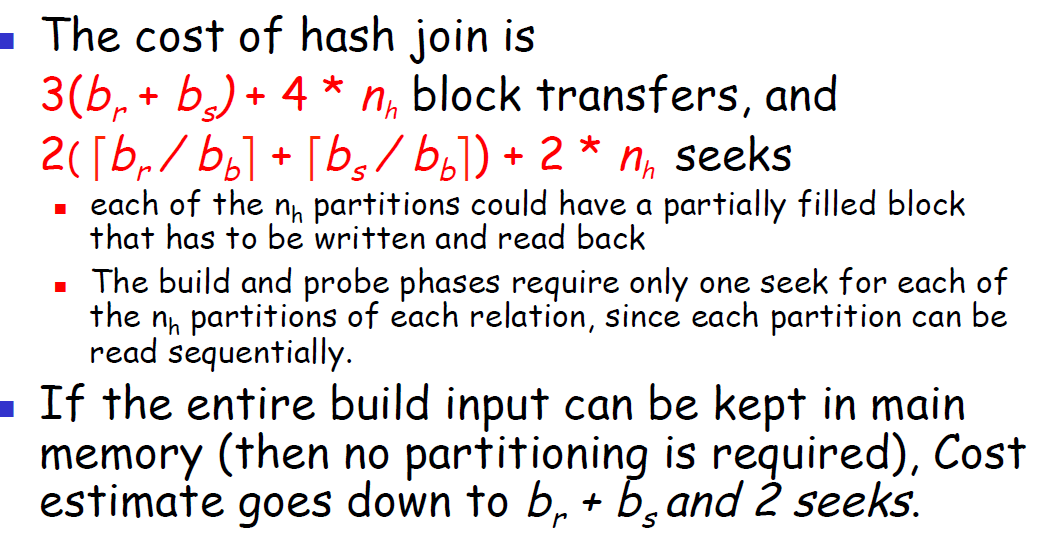
*并归连接：(在计算未排序好的关系是，要先计算已经排序好的再加上未排序好的)*

如果关系已排序：则块传输次数是:br+bs. 磁盘搜索次数是: ⌈br/bb⌉+⌈bs/bb⌉ （bb是分配的缓冲块，最好情况是1­­,下面的M是并归段）

如果关系未排序：先计算进行排序时需要的递归次数：⌈logm-1(br/M) ⌉。再计算进行块传输次数: br\*(2\*⌈log取下整[M/bb]-1 (br/M) ⌉+1)。磁盘的搜索次数是:

2\*⌈br/M⌉+⌈br/bb⌉\*(2\*⌈log取下整[M/bb]-1(br/M)⌉-1)

散列连接：



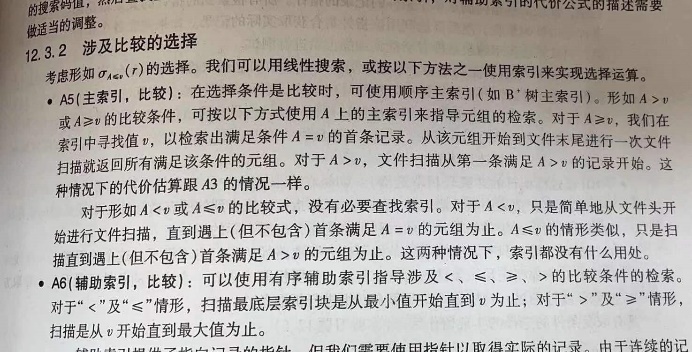
计算需要多少次block transfer 和 多少次seek

如果是查询单个项liner scan 则 cost estimate = brblock transfer + 1seek

If selection is on a key attribute, can stop on finding record(linear search, Equality on key):

Average cost = (br/2) block transfer + 1seek

如果查询使用binary search: cost of locating the first tuples by a binary search on the block

描述如何使用B+数在一个属性上的查找

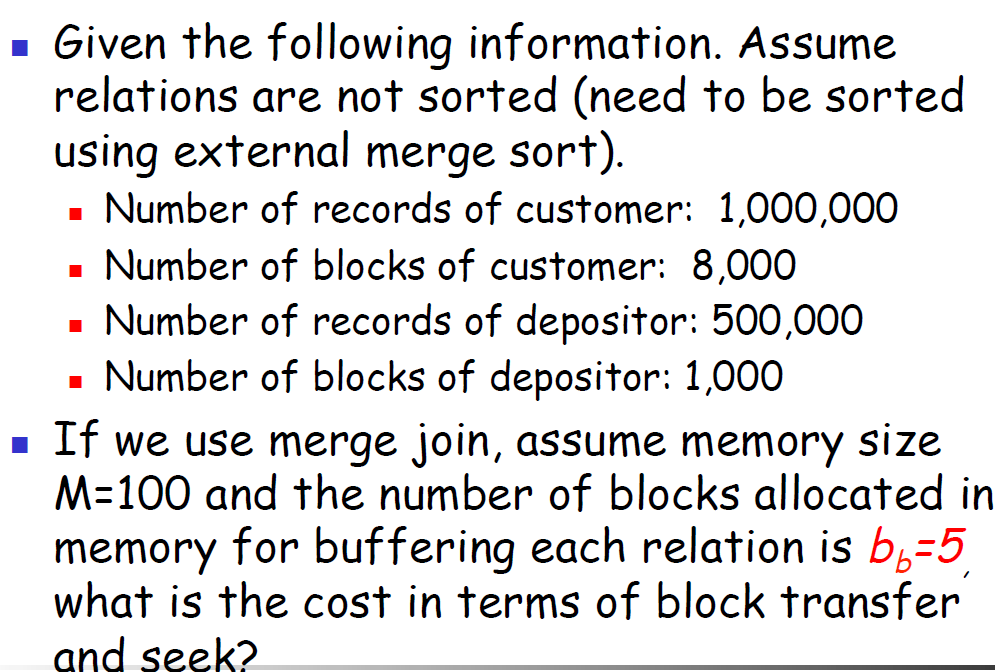
用merge algorithm 算法计算需要多少个block 和多少次transfer

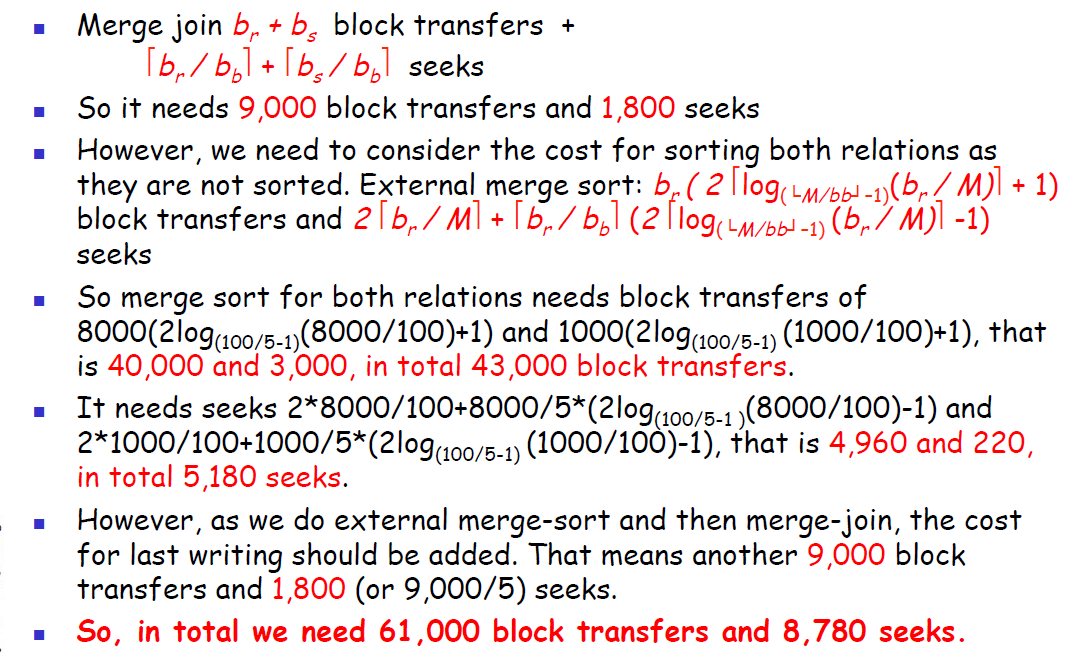
如果排序好

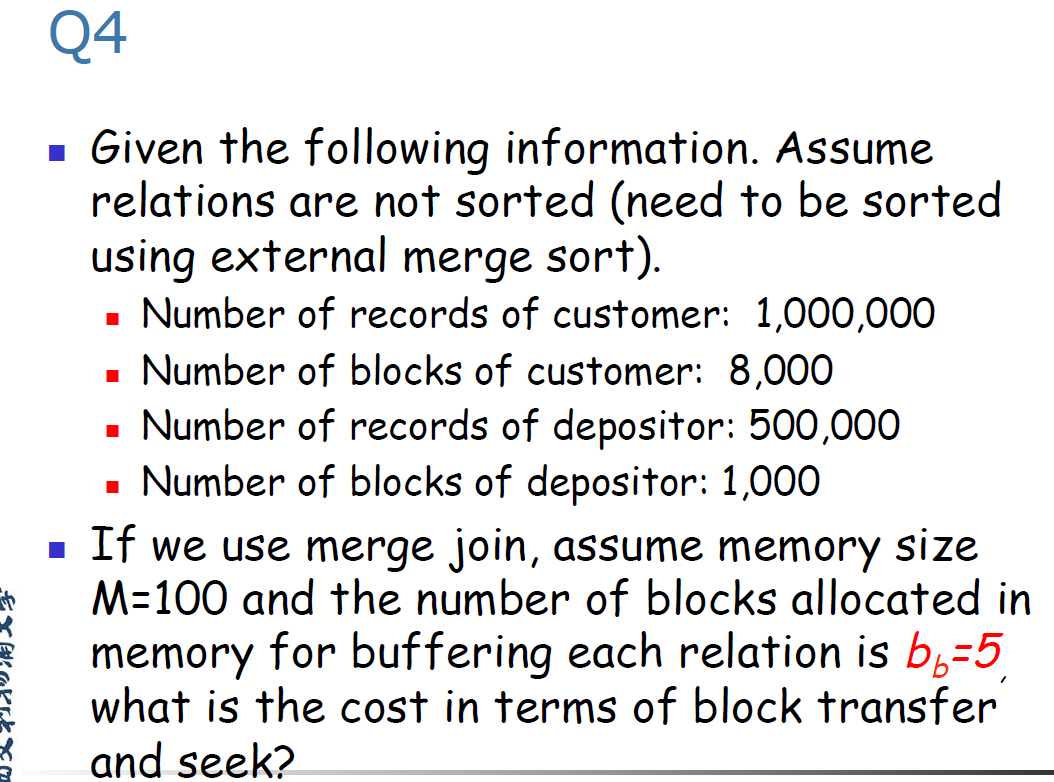
1. 直接用公式br+bs计算transfer和公式⌈br/bb⌉+⌈bs/bb⌉计算seek。(bb是缓冲区)

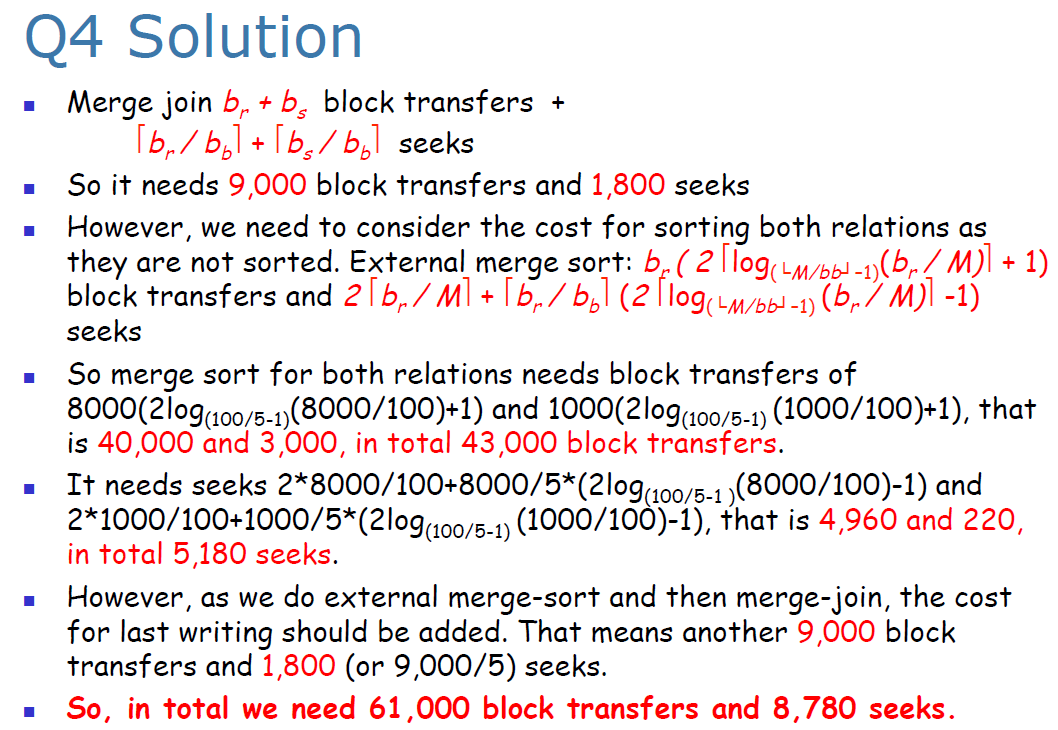
如果未排序：

1. 先用上面的公式计算一遍以后用。
2. 再用公式br\*(2\*⌈log取下整[M/bb]-1 (br/M) ⌉+1)计算transfer次数(计算两次)，用公式2\*⌈br/M⌉+⌈br/bb⌉\*(2\*⌈log取下整[M/bb]-1(br/M)⌉-1)计算seek次数(计算两次)。 注意 两个关系都得计算，也就是一共需要计算4次。
3. 第三步把2得出来的结果加上两倍1的结果就是总结过，分开算哦。

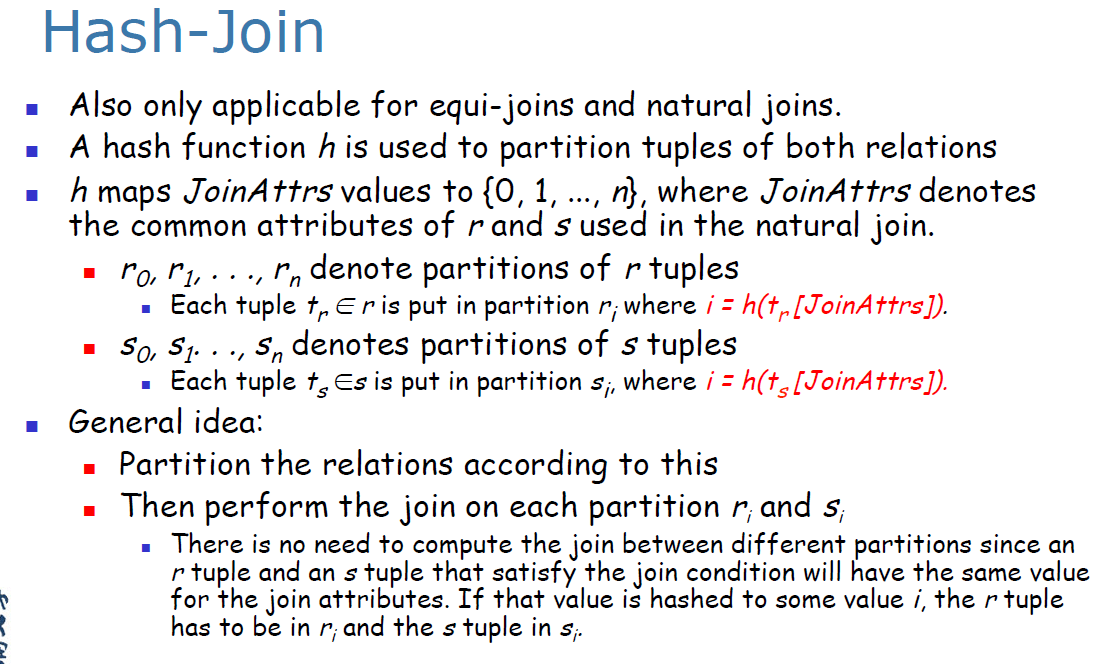
例：







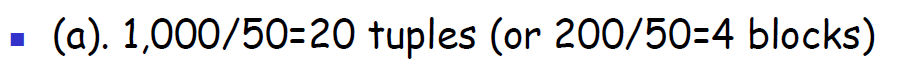
当发现题目中给了index nested loop join 就得用nr\*c+br去算，一般情况下树高不是三就是四。只用到有B树的那个relation，如果两个都有B树就用少的那个去算。



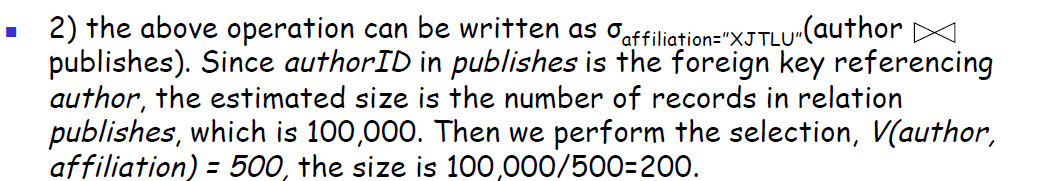
## Q3

**estimated 一个查询的size：**

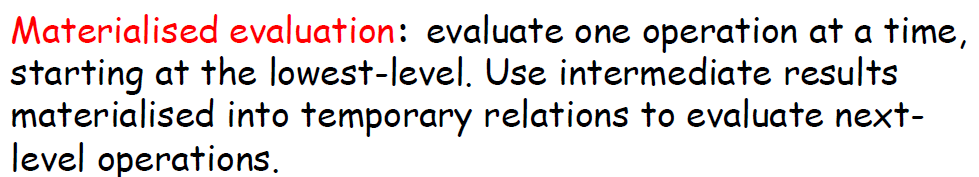
一般考这个题提干中都会带V(A,r)这意思是关系中属性A中出现非重复值的个数，如果A是关系中的主码，则V(A,r)等于n

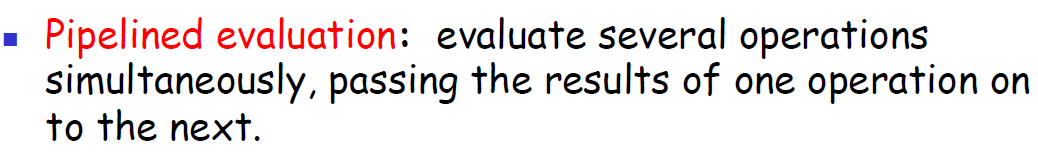
解用tuple数/V（or block/r）

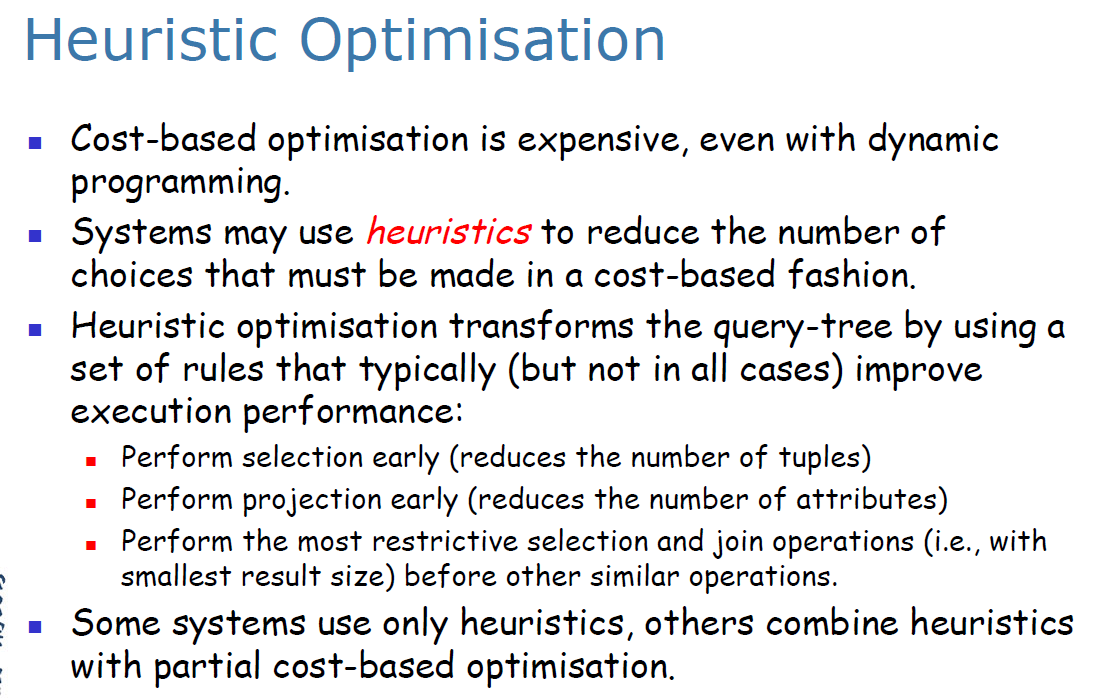
**estimated 一个operation 的size：**

不是很懂，就感觉是把给的关系先化简一下，转化成可以往V靠的关系，然后再用tuple数/v

给你一个关系代数 然后计算transfer时有多少个block需要被evaluate, 使用indexed nested loop join 算法







## Q4

描述事务四大特性：

Atomicity: Either all operations of the transaction are properly reflected in the database or none are.

Consistency: Execution of a transaction in isolation preserves the consistency of the database.

Isolation: Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate

transaction results must be hidden from other concurrently executed transactions.

Durability: After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

事务：

事务隔离性存在隔离级别，理论上隔离级别包括4个：

第一级别：读未提交(read uncommited)

对方事务还没有提交，我们当前事务可以读到对方未提交的数据。

读未提交脏数据(Dirty Read)现象：表示读到了脏得数据。

第二级别：读已提交(read commited)

对方事务提交之后得数据我方可以读取到。

读已提交存在的问题是：不可重复读

第三级别：可重复读(repeatable read)

这种隔离级别解决了：不可重复读问题

第四级别：序列化读/串行话化读

解决了所有问题。

效率低，需要事务排队。

可串行化：优先图

如果满足以下三点条件的任意之一则Ti🡪Tj

在Tj执行read(Q)之前，Ti执行write(Q)

在Tj执行write(Q)之前，Ti执行read(Q)

在Tj执行write(Q)之前，Ti执行write(Q)

日志文件的内容：

[start,T]: 事务的开始

[write, T, X, old\_value,new\_value]: 事务T更新X

[commit, T]: 事务提交

[abort,T]: 事务撤销

事务故障恢复：事务在运行至正常终点之前被终止

恢复方法：又恢复子系统应利用日志文件撤销(UNDO)此事务以对数据库进行修改。

事务故障的恢复由系统自动完成。不需要用户干预。

事务故障的恢复步骤

1：反向扫描日志文件(即从最后向前扫描日志文件)，查找该事务的更新操作。

2：对该事务的更新操作执行逆操作。将日志文件记录中’’更新前的值‘’(Before Image BI)写入数据库。若是修改操作则用BI代替AI(After Image)

并发控制：

排他锁(X锁)

又称写锁，若事务T对数据对线A加上X锁，则只允许T读取和修改A，其他任何事务都不能再对A加任何类型的锁，直到T释放A上的锁

共享锁(S锁)

又称为读锁，若事务T对数据对象A加上锁S，则其他事务只能再对A加S锁，而不能加X锁，直到T释放A上的S锁

一级封锁协议：在写之前加上X锁，写之后释放X锁

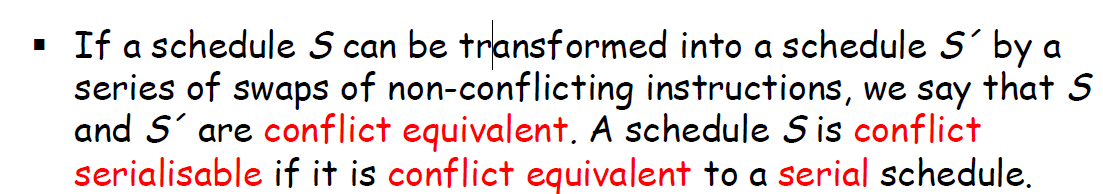
二级封锁协议：在读之前加上X锁，读之后释放X锁，此时的问题是两个读之间有空窗期

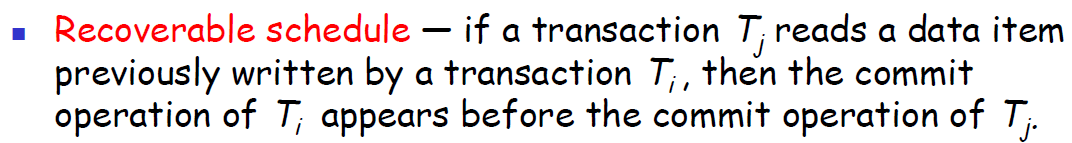
三级封锁协议：在读之前加上S锁，读之后释放S锁，在写之前加上X锁，写之后释放X锁

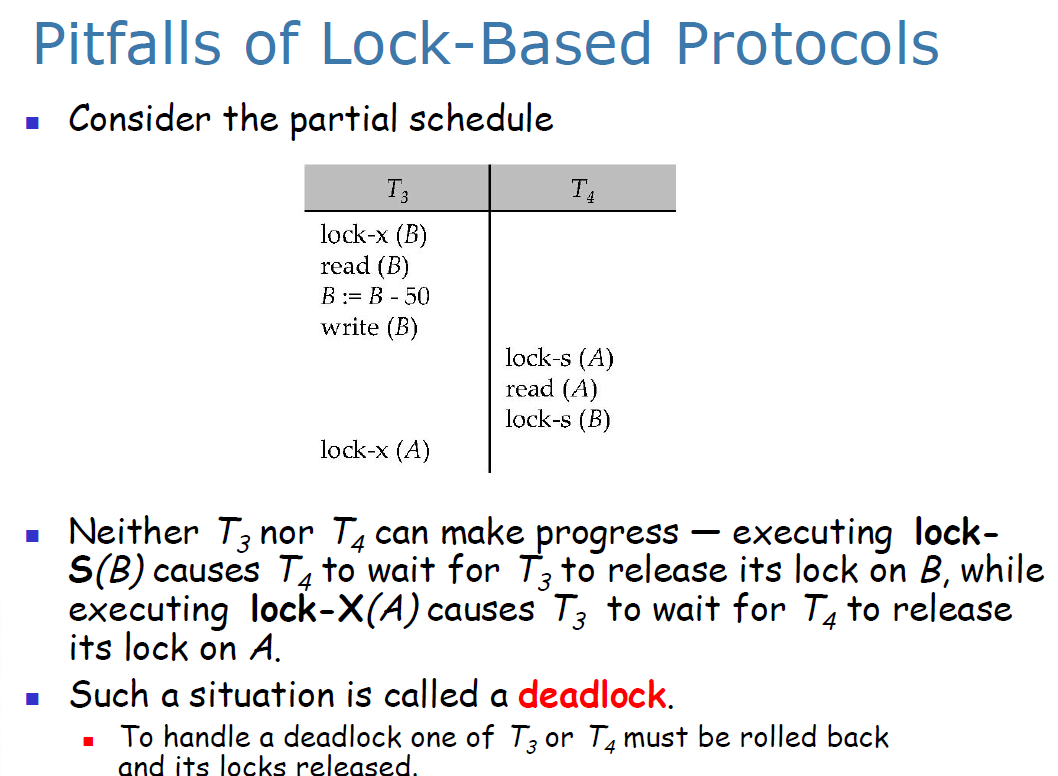
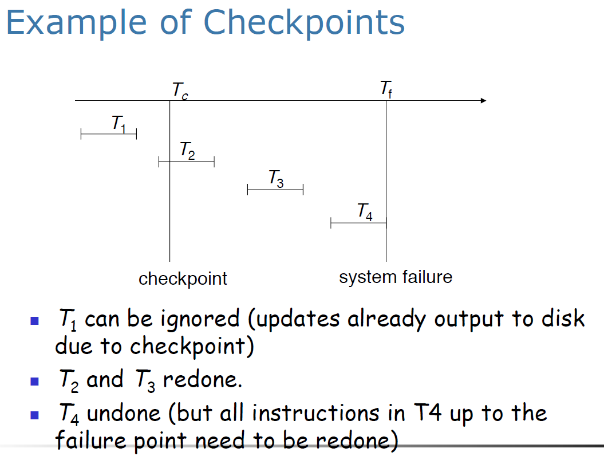
找时间做一下w5a的example

看lec6a的可恢复性，顺便看ttlQ3

What is meant by ‘a schedule is conflict serializable(可序列化)’







在checkpoint 之前已经commit的也要redone

## Q5：

Object-Relational Data Models

他是继承于关系数据模型

Complex Data type