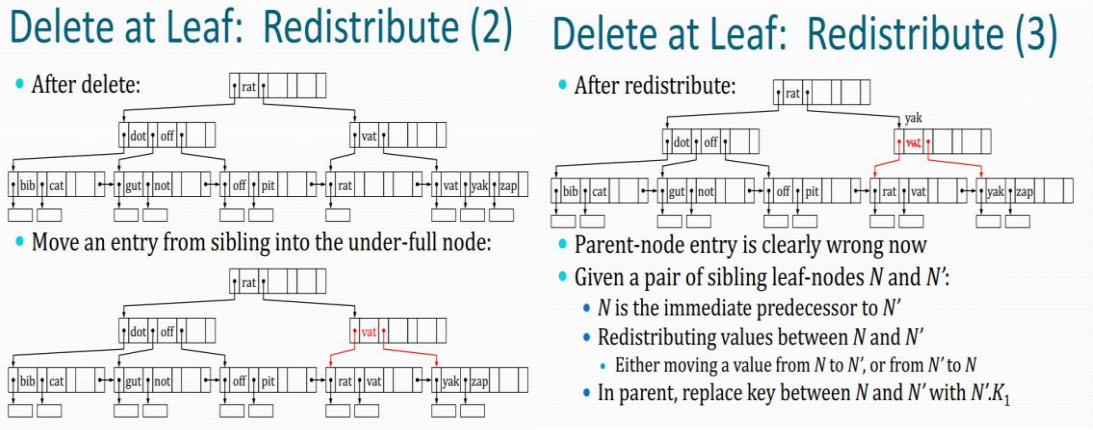
1. The student records are sequentially ordered by s!D and the "clustered disk organisation" strategy is used. This means that, for **each student record, the 5 module records** also reside in the same block.
   Also, assume that a record **does not span over more than one block**
   潜台词:
   先使用 block size 计算每个 block 能存多少 tuple，取最小（四舍五舍），然后再用 tuple 的总数去计算能放多少个 block，取最大（四入五入），这样算出来就不会有一个 record 分开存在两个 block 里面。
   而且根据表格之间的关系也有相应的包含，一个表 reference 另一个表…
2. 注意 B+树 split 算法的不同，以老师 ppt 为主，分裂叶节点时插入父节点的 key 编号是 n/2(向上取整) +1。但是分裂非叶节点插入父节点的 key 编号是 n/2(向上取整)。
3. 注意哈希 index 是否是要使用 overfitting 的情况，只有当哈希值完全相同时才需要使用 overfitting，练习题里 overfitting 的情况很多所以容易误用 overfitting，实际情况要先注意 i_j 还能不能再分了。
4. 注意题目中询问的是 can be evaluated in the most cost effective way by using a xxx and xxx，还是让计算出一个实际的数值，不要把对比题当作需要计算出实际数据的题目。
5. **Assume that the memory can only hold one block for each relation.**
   这句话是帮助判断 blocked nested loop join algorithm 的，因为主存只能放一个 block 所以每一次在 outer relation 上计算 inner relation 的时候都要重新传一个 block 运算，直接套公式就行。
6. 注意使用 merge join algorithm 的时候两个 relation 是否 have been sorted。
7. 注意观察要计算的是 estimated size 还是 block transfer num，如果是前者基本上和使用什么 Join 的算法关系不大，只要知道两个 Relation 相交之后的属性的关系以及两个关系的 tuple 数，就可以计算 size 了(block 不要随意计算因为你不知道这两者的单独一个 record 多少比特 不要随意换算)。
8. 在第一层的选择使用 pipeline 没有意义 还会被陈江寒嘲笑 不要忘记从磁盘读取关系是需要 Block transfer 的 不能在这里用 Pipeline
9. 算法中的 block transfer 本身就包括读取块的 cost 不用额外加。
10. 冲突可串行化…P549
11. B+树删掉一个项重新合并之后得到的结果，如果 N'在 N 前面，那么完成分配之后还需要有一次重新分配以符合父节点左边的 pointer 定位的子节点值小于父节点，右边的子节点包括父节点值本身和比其大的值（也有可能这个父节点值没有对应的 index，但是在删除后改节点值被保留下来打工了）。



12. B+树中 N 为高度， n 为每个节点的 Pointer 数量 不要搞混了。。。
13. 常用公式整理：

Confidence and **Support** of Rules:

Confidence and Support of Rules

- We prune the set of all possible **association rules** using two measures:
- **Support** of a rule:
  - **X => Y has support s if P(X, Y) = s**
  - (#of transactions contain both X and Y/total of transactions)
- **Confidence** of a rule:
  - **X => Y has confidence c if P(Y|X) = c**
  - (#of transactions contain X and Y/# of transactions contain X)
- **Support of a co-occurrence XY**:
  - **XY has support s if P(X, Y) = s**
  - (#of transactions contain X and Y/total of transactions)
  - Same as X => Y

- **Support** is the number of transactions that include items in the {A} and {B} parts of the rule as a percentage of the total number of transactions.

$$\textbf{Support} = \frac{(\mathbf{A} + \mathbf{B})}{\textbf{Total}}$$

Interpreted as: *Fraction of transactions that contain both A and B.*

- **Confidence** - *How often items in B appear in transactions that contain A only.*

$$\textbf{Confidence} = \frac{(\mathbf{A} + \mathbf{B})}{\mathbf{A}}$$

- **Support of a co-occurrence (lift)** - interpreted as: *How much our confidence has increased that B will be purchased given that A was purchased.*

$$\textbf{Lift} = \left(\frac{\left(\frac{(\mathbf{A} + \mathbf{B})}{\mathbf{A}} \right)}{\left(\frac{\mathbf{B}}{\textbf{Total}} \right)} \right)$$

Xi'an Jiaotong-Liverpool University 西交利物浦大学

17

选择算法代价估计：

	Algorithm	Cost	Reason
A1	Linear Search	$t_S + b_r * t_T$	One initial seek plus b_r block transfers, where b_r denotes the number of blocks in the file.
A1	Linear Search, Equality on Key	Average case $t_S+(b_r/2) * t_T$	Since at most one record satisfies the condition, scan can be terminated as soon as the required record is found. In the worst case, b_r block transfers are still required.
A2	Clustering B^+-tree Index, Equality on Key	$(h_i + 1) * (t_T + t_S)$	(Where h_i denotes the height of the index.) Index lookup traverses the height of the tree plus one I/O to fetch the record; each of these I/O operations requires a seek and a block transfer.
A3	Clustering B^+-tree Index, Equality on Non-key	$h_i * (t_T + t_S) + t_S + b * t_T$	One seek for each level of the tree, one seek for the first block. Here b is the number of blocks containing records with the specified search key, all of which are read. These blocks are leaf blocks assumed to be stored sequentially (since it is a clustering index) and don't require additional seeks.
A4	Secondary B^+-tree Index, Equality on Key	$(h_i + 1) * (t_T + t_S)$	This case is similar to clustering index.
A4	Secondary B^+-tree Index, Equality on Non-key	$(h_i + n) * (t_T + t_S)$	(Where n is the number of records fetched.) Here, cost of index traversal is the same as for A3, but each record may be on a different block, requiring a seek per record. Cost is potentially very high if n is large.
A5	Clustering B^+-tree Index, Comparison	$h_i * (t_T + t_S) + t_S + b * t_T$	Identical to the case of A3, equality on non-key.
A6	Secondary B^+-tree Index, Comparison	$(h_i + n) * (t_T + t_S)$	Identical to the case of A4, equality on non-key.

Figure 15.3 Cost estimates for selection algorithms.

B+ tree height:

- If there are K search-key values in the file
  - The B^+-tree height is no more than $\lceil \log_{\lceil n/2 \rceil}(K) \rceil$.

B+ 树特性：(一个非叶节点最少需要几个 pointer，一个叶节点最少需要几个 key)

In a B+-Tree,

- n (or sometimes N) is the number of pointers in a node; pointers: P1, P2, ...Pn
- Search keys: K1 < K2 < K3 < . . . < Kn-1
- All paths (from root to leaf) have same length
- Root must have at least two children
- In each non-leaf node (inner node), more than 'half' ($\geq\lceil n/2\rceil$) pointers must be used
- Each leaf node must contain at least $\lceil (n-1)/2 \rceil$ keys

Nested-Loop Join Cost:

In the worst case, if there is enough memory only to hold one block of each relation, n_r is the number of tuples in relation r, the estimated cost is

$n_r * b_s + b_r$ block transfers, plus

$n_r + b_r$ seeks

If the smaller relation fits entirely in memory, use that as the inner relation.

- Reduces cost to $b_r + b_s$ block transfers and 2 seeks

But in general, it is much better to have the smaller relation as the outer relation

The choice of the inner and outer relation strongly depends on the estimate of the size of each relation.

Block Nested-Loop Join Cost:

Block Nested-Loop Join Cost

- Worst case estimate: $b_r * b_s + b_r$ block transfers and $2 * b_r$ seeks
  - Each block in the inner relation s is read once for each *block* in the outer relation (instead of once for each tuple in the outer relation).
- Best case (when smaller relation fits into memory): $b_r + b_s$ block transfers plus 2 seeks.

Indexed Nested-Loop Join: (c 的值见上表选择算法代价估计)

Cost of the join: $b_r + n_r * c$ block transfers and seeks

- Where c is the cost of traversing index and fetching all matching s tuples for one tuple in r
- c can be estimated as cost of a single selection on s using the join condition (usually quite low, when compared to the join)

Merge-Join cost:

Thus the cost of merge join is (where b_b is the number of blocks in allocated in memory for each relation):

$b_r + b_s$ block transfers +

$\lceil b_r / b_b \rceil + \lceil b_s / b_b \rceil$ seeks

- Plus the cost of sorting if relations are unsorted.

Merge Sort (Cost analysis)：

External Merge Sort (Cost analysis)

Assume relation is stored in b_r blocks, M is the memory size, so the number of run file b_r/M.

Buffer size b_b (read b_b blocks at a time from each run and b_b blocks for writing).

Cost of Block Transfer

- Total number of merge passes required: $\lceil \log_{\lfloor M/b_b \rfloor - 1}(b_r/M) \rceil$. Each time can merge $\lfloor (M-b_b)/b_b \rfloor$.
- Block transfers for initial run creation as well as in each pass is $2b_r$ (read/write all b_r blocks)
  - for final pass, we don't count write cost
  - Thus total number of block transfers for external sorting:
    $2b_r + 2b_r \lceil \log_{\lfloor M/b_b \rfloor - 1}(b_r / M) \rceil - b_r = b_r (2 \lceil \log_{\lfloor M/b_b \rfloor - 1}(b_r / M) \rceil + 1)$

Cost of seeks

- During run generation: one seek to read each run and one seek to write each run
  - $2 \lceil b_r / M \rceil$
- During the merge phase
  - Need $2 \lceil b_r / b_b \rceil$ seeks for each merge pass
  - Total number of seeks:
    $2 \lceil b_r / M \rceil + \lceil b_r / b_b \rceil (2 \lceil \log_{\lfloor M/b_b \rfloor - 1}(b_r / M) \rceil - 1)$

Cost of Hash-Join:

The cost of hash join is

$3(b_r + b_s) + 4 * n_h$ block transfers, and

$2(\lceil b_r / b_b \rceil + \lceil b_s / b_b \rceil) + 2 * n_h$ seeks

- each of the n_h partitions could have a partially filled block that has to be written and read back
- The build and probe phases require only one seek for each of the n_h partitions of each relation, since each partition can be read sequentially.

$n_h = (b_s/M)*f$　ps: (b_s/M)向上取整

14. Catalog Information for Cost Estimation

数据库目录信息，表达式估计相关计算：第六版中文 p335

n_r: number of tuples in a relation r.

b_r: number of blocks containing tuples of r.

l_r: size of a tuple of r.

f_r: blocking factor of r. i.e., the number of tuples of r that fit into one block.

$V(A, r)$: number of distinct values that appear in r for attribute A; same as the size of $\prod_A(r)$.

If tuples of r are stored together physically in a file, then: $b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$

选择规模估计：Estimation of the Size of Selection

$\sigma_{A=V}(r)$

- $n_r / V(A,r)$: number of records that will satisfy the selection
- Equality condition on a key attribute (primary key): *size estimate* = 1

$\sigma_{A\leq V}(r)$ (case of $\sigma_{A\geq V}(r)$ is symmetric)

- Let c denote the estimated number of tuples satisfying the condition. Let min(A,r) and max(A,r) denote the lowest and highest values for attribute A.
- If min(A,r) and max(A,r) are available in catalog
  - c = 0 if v < min(A,r)
  - c = $n_r \cdot \frac{v - \min(A,r)}{\max(A,r) - \min(A,r)}$
- If histograms available, can refine above estimate
- In absence of statistical information c is assumed to be $n_r / 2$.

连接规模估计：Estimation of the Size of Joins　第六版中文 p338

The Cartesian product $r \times s$ contains $n_r . n_s$ tuples; each tuple occupies $s_r + s_s$ bytes.

If $R \cap S = \varnothing$, then $r \bowtie s$ is the same as $r \times s$.

If $R \cap S$ is a key for R, then a tuple of s will join with at most one tuple from r

- therefore, the number of tuples in $r \bowtie s$ is no greater than the number of tuples in s.

If $R \cap S$ is a foreign key in S referencing R, then the number of tuples in $r \bowtie s$ is exactly the same as the number of tuples in s.

- The case for $R \cap S$ being a foreign key referencing S is symmetric.

If $R \cap S = \{A\}$ is not a key for R or S.
If we assume that every tuple t in R produces tuples in $R \bowtie S$, the number of tuples in $R \bowtie S$ is estimated to be:

$$\frac{n_r * n_s}{V(A,s)}$$

If the reverse is true, the estimate obtained will be:

$$\frac{n_r * n_s}{V(A,r)}$$

The lower of these two estimates is probably the more accurate one.

15. State two key characteristics of object-relational models, compared to traditional relational databases…

RDBMS and OODBMS are database management systems. RDBMS uses tables to represent data and their relationships whereas OODBMS represents data in form of objects similar to Object Oriented Programming.

Following are the important differences between RDBMS and OODBMS.

Sr. No.	Key	RDBMS	OODBMS
1	Definition	RDBMS stands for Relational DataBase Management System.	OODBMS stands for Object Oriented DataBase Management System.
2	Data Management	Data is stored as entities defined in tabular format.	Data is stored as objects.
3	Data Complexity	RDBMS handles simple data.	OODBMS handles large and complex data.
4	Term	An entity refers to collection of similar items having same definition.	An class refers to group of objects having common relationships, behaviors and properties.
5	Data Handling	RDBMS handles only data.	OODBMS handles both data and functions operating on that data.
6	Objective	To keep data independent from application program.	To implement data encapsulation.
7	Key	A primary key identifies in object in a table uniquely.	Object Id, OID represents an object uniquely in group of objects.