WRITTEN ASSIGNMENT 2

Onur Can ÜNAL
2095966

(1)

a) It is possible to evaluate this query with an index-only plan. Since there is a composite search key (age, grade), there is no need to check actual records for grouping by age. WHERE clause also can be achieved by index files. After grouping count(*) and MIN(grade) can be obtained without accessing actual records.

b) It is not possible to evaluate this query with an index-only plan. Since the composite key (age, grade), the index files include age and grade. Any query that needs only those fields with or without aggregations can be achieved with an index only plan. However, in WHERE clause, there is an equality search for gender = 'Female'. As index files don't include gender field, it has to look actual records that are in leaf nodes of B+ tree. Thus, this query cannot be achieved by an index-only plan with the composite search key (age, grade).

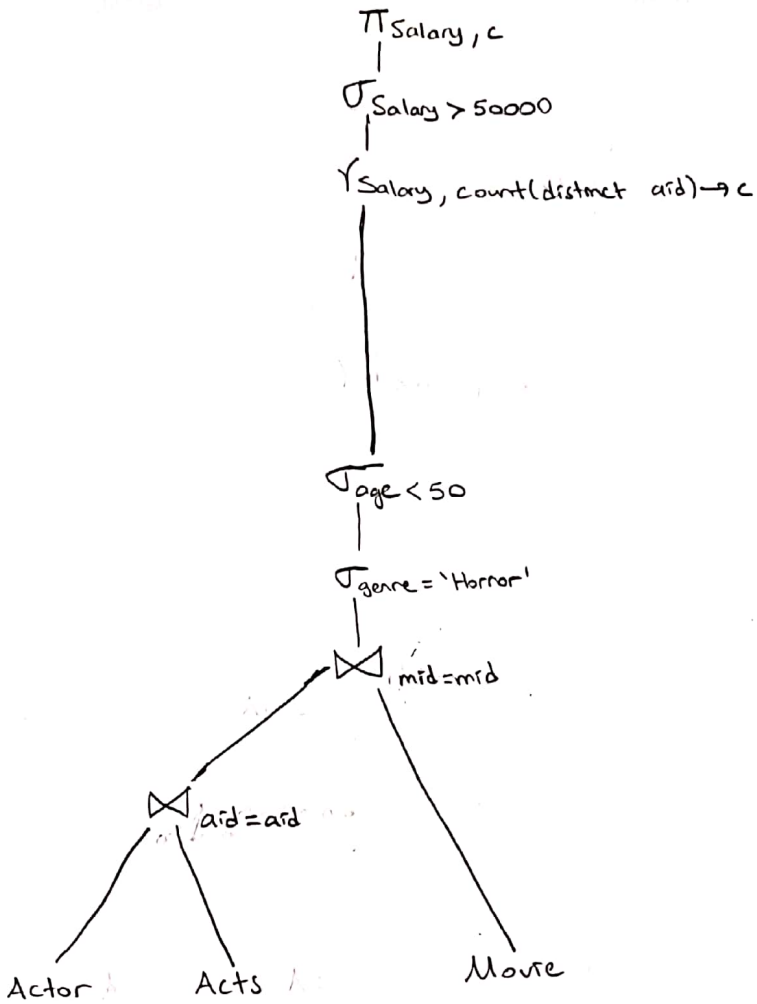
(2) a) Using a hash index on attribute R.A is the best choice for this query since it is a point query (equality search query). By using hash function with the input A = 500.000, it takes constant time, and so it is likely to have the least cost.

b) This query is range query. It wants the records that have A field value from 1 to 19.999. B+ Tree indexes are the best choice for range queries. Since in the clustered index records close in index are close in data while in the unclustered index records close in index may be far in data, using a clustered B+ tree index is the best choice here. With the clustered B+ index, DBMS starts searching the key value from the root. It then proceeds down the leaf accordingly. Then, since the values are sorted and linked to each other in the leaf, it traverses them sequentially. Thus, for the range queries, using a clustered B+ index is the best choice.

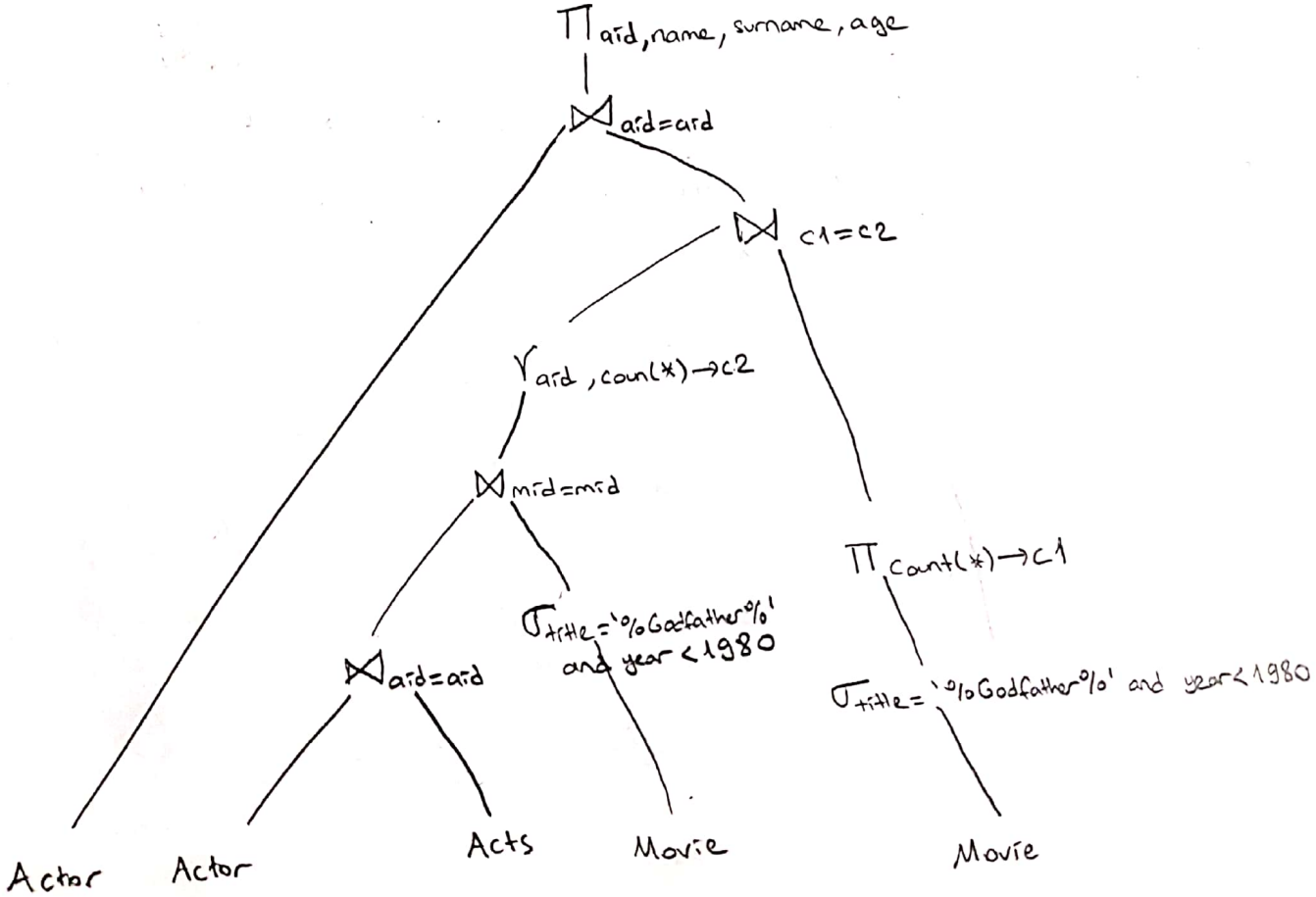
c) This query is also a range query. It wants the records that have A field value from 20.001 to 20.009. Thus, as in part b), using a clustered B+ index is likely to have the least cost.

d) This is an inequality search query. Although it looks similar to part a), it is different. DBMS needs to check all records for giving the results. Using unclustered B+ tree index or hash index are the worst choices since there is a need to check all records separately, these would be 10.000.000 I/Os. Using a clustered B+ tree index is a better choice, but these would be an overhead because of searching indexes in addition to the checking records. Using a heap file is likely to have least cost, since all the pages loaded in the memory and each is checked one by one. With the heap file cost would be the number of pages 1.000.000 I/Os.

(3) a)



b)



(4)

a) $Cost = B(R) + B(R)B(S)/(M-2)$

$B(R) = 20.000/10 = 2.000$ blocks (pages)
$B(S) = 5.000/10 = 500$ blocks (pages)
$M = 42$ pages

$Cost = 2.000 + 2.000 \times 500/40$

$= \underline{27.000 \text{ I/Os}}$ ✓

b) $Cost = B(S) + B(S)B(R)/(M-2)$

By getting values from part a):

$Cost = 500 + 500 \times 2.000/40$

$= \underline{25.500 \text{ I/Os}}$ ✓

c) $B(R) = 2.000$ blocks
$B(S) = 500$ blocks

Since $[B(R)]^2 > 42^2$, we can not use 2 pass sort-merge join. We need at least 3 pass sort-merge join. Let's use 3 pass sort-merge join. Firstly, all of 42 pages that are in the memory are used to read and sort the pages of R relation, and then S relation. All of the pages of those relations are read, sort and written to the disk with 42 page at a time. At the end, in the disk, there will be 48 runs of R ($\lceil 2000/42 \rceil$), and 12 runs of S ($\lceil 500/42 \rceil$). Since $48+12 > 42$, we can not merge them, directly. We need to merge runs of R within each other, firstly. At merge step, there is 1 page for output, and there are 41 pages to hold the sorted pages of R. By this way, 41 runs of R are merged to 1 bigger chunk with the size of 42×41 page. Since at the beginning, we had 60 chunks, at the end we will have 2 chunks that are sorted. At the merge step between R and S, 2 memory pages are allocated for the 2 chunks of R, and 12 memory pages are allocated for the 12 chunks of S, and 1 page is allocated for the output to be merged. Each page of all of the chunks are, read one by one, merged, and written to the disk. This concludes merge-sort process.

Total Cost = $\underbrace{2B(R) + 2B(S)}_{\text{read, sort, and write initially}} + \underbrace{2B(R)}_{\text{read+merge and sort the chunks of R}} + \underbrace{B(R) + B(S)}_{\text{merging process}} = 5B(R) + 3B(S) = 5 \times 2000 + 3 \times 500$

$= \underline{11.500 \text{ I/Os}}$ ✓

d) Since $B(R) = 2.000$ and $B(S) = 500$, and $\min(2.000, 500) <= 42^2$, we can use a hash join with 2 pass.

All of 42 pages that are in the memory used, firstly. One of them is used for input buffer and the rest $(M-1)$ 41 are used for buckets to hash into. Firstly, all of 500 pages that are in relation S is read one page at a time and hash into 41 buckets. When a bucket fills up, the page that is in that bucket is flushed into the disk. At the end, S relation is got in the disk with 41 splitted buckets. Then, R relation is also read similarly. After then, all of 41 partitions of R is read one by one, and they are inserted into the created hash table in the memory. At this step, there 1 input buffer with the size of one page, 1 output buffer with the size of one page and hash table with the size of 40 pages. Then, scan the pages of S one by one into the input buffer, and match with the hash table. Repeating this process for all the buckets ends the process.

Total Cost = $3B(R) + 3B(S) = 3 \times 2000 + 3 \times 500 = \underline{7,500 \text{ I/Os}}$

e) Two relations are joined with an index by iterating over R and for each tuple in R, fetch corresponding tuples from S. Both joins with clustered and unclustered index works similarly, but the main difference between them is that while relation is read one page at a time and fetching tuples are read one page at a time, since S is sorted if it has clustered index, the likelihood of a page miss is far lower than that of having unclustered index. However, in this question, since attribute b is primary key for S, the result will be same. (because b is primary key each time 1 tuple will be fetched)

i. Clustered index cost = $B(R) + T(R)B(S)/V(S,b)$

$= 2.000 + 20.000 \times \underbrace{500/5.000}_{1} = 22.000$ I/Os.

ii. Unclustered index cost = $B(R) + T(R)T(S)/V(S,b)$

$= 2,000 + 20,000 \times 5.000/5.000 = 22,000$ I/Os.

(5) a) $(m_{10} + m_{11} + m_{12}) \times \frac{t_{swim}}{N}$

b) In this estimate, it is assumed that each of the counts that is t_{swim}, t_{mitten}, t_{garden} are uniformly distributed over all months. We assumed that for each month the ratio of t_{swim} to all records are the same. However, this cannot imitate the real life, because in the 10th, 11th and 12th months of a year, we expect that the numbers of swim clothes are the lowest numbers of a year. We expect similar behaviours for other types of clothes, also. Thus, this estimate may be incorrect.