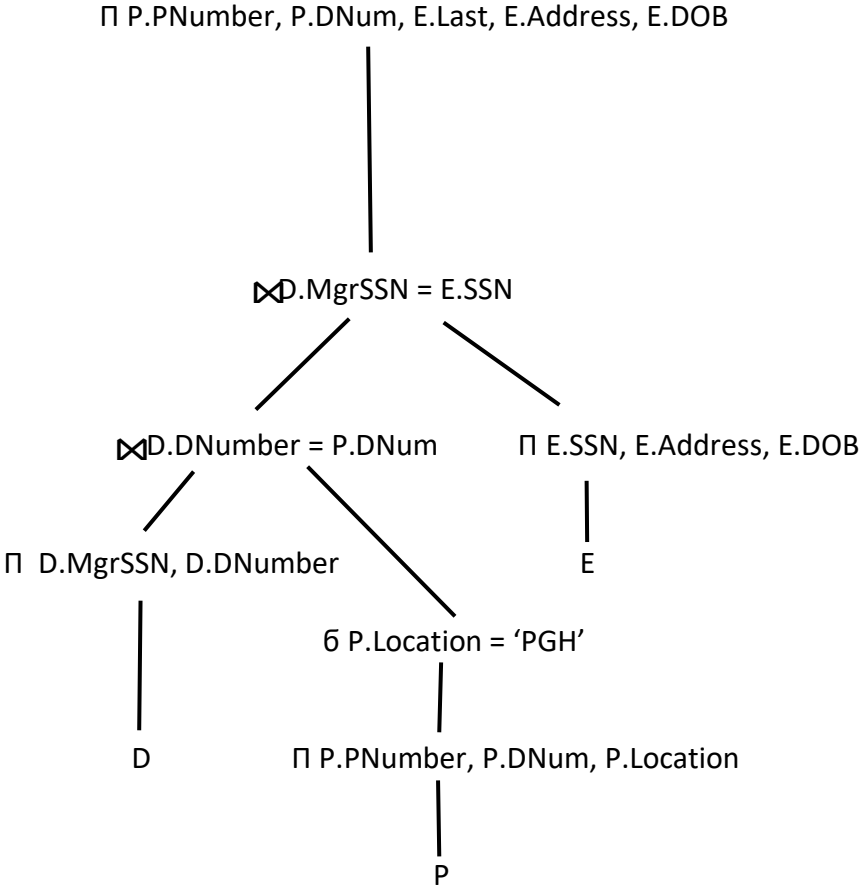
Assumptions for all sections:
DNumber is the primary key of relation D, selectivity = 1/2500
PNumber is the primary key of relation P, selectivity = 1/5000
SSN is the primary of relation E, selectivity = 1/10000
DNum is a foreign key of P referencing D
MgrSSN is a foreign key of D referencing E
1 page is one block-size

1.



Implementation:
Projections in the three heap files also cost one loop over the table since all the projections involve primary keys as attributes. The join operations require nested loop join, since the tuples are unsorted and have no access methods. The selection operation involves a string comparison on heap file so it costs linear runtime.
Costs:
At the leaves level we project the listed attributes from D and P, numbers of disk reads here are B_p + B_dep = 500 + 1000 = 1500 block reads. We can read in all the blocks in one loop since the projection involves primary keys. These reads require 1500 / buffer_size = 1500 / 20 =

75 runs. After retrieving the required attributes within memory, we must write the same # of the blocks back to disk as two intermediate tables. Thus, the first level I/O cost is 3000.

Then we do the selection operation on P's intermediate table. We do this first to reduce the subsequent I/O. We use buffer_size - 1 # of pages to read in all blocks leave one page for result. This costs 1000 (P's cardinality) block reads (1000 / 19 = 27 runs) and 500 result block writes (half of the rows have 'PGH'), thus the I/O cost is 1500 so far. We can perform projection on E here as well. Then we perform projection on E which costs a single loop, causing E_b * 2 = 2000 * 2 (read + write) = 4000. Thus the total I/O cost is 5500 at this level.

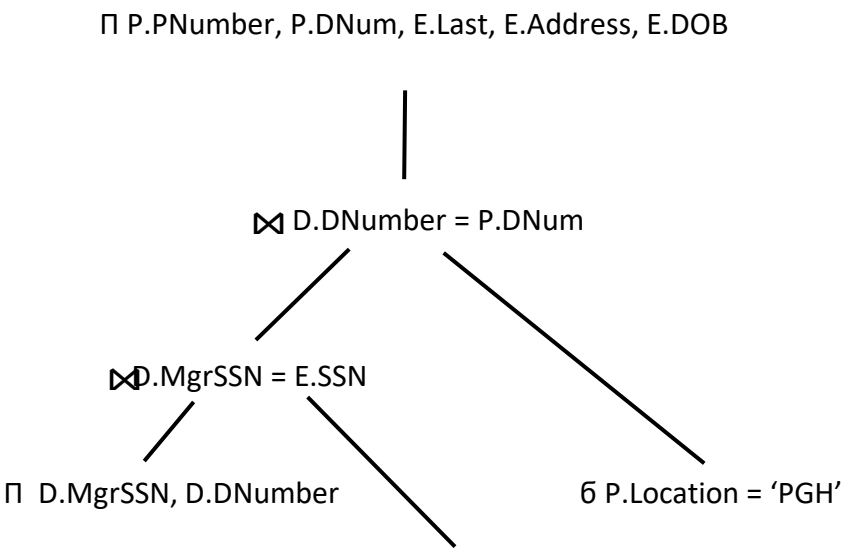
Then we do the join operation with D.DNumber = P.DNum. Using nested Loop join, we choose either intermediate table as outer loop (both have 500 as cardinality) We use the 20 - 2 pages in the buffer to read outer table blocks and the other two for reading inner table blocks and writing results. This will result in 500 + (500 / 18) * 500 = 14389 block reads in total. We need to write 500 blocks (joined by primary key) back to disk as a intermediate table. The total I/O cost is 14389 + 500 = 19389

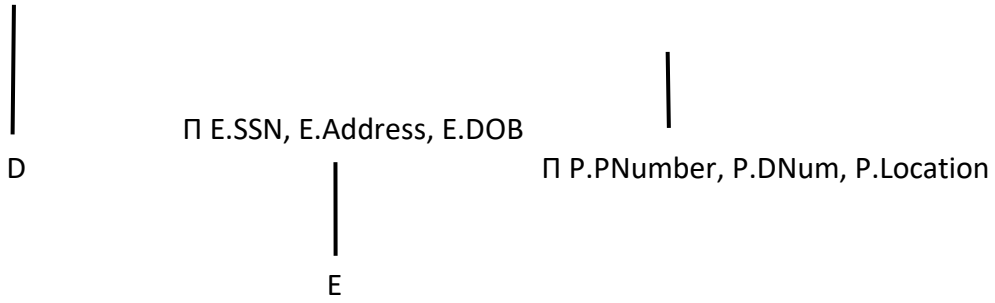
Next up, we have a equi-join on the last two intermediate table. Again same nested loop join method and use the smaller table for outer loop, resulting in 500 + (500 / (20 - 2)) * 2000 = 56056 block reads and 500 result block writes (joined by primary key). the total I/O costs is 56056 + 500 = 56556

Finally, we perform projection on the last intermediate table. Since the initial intermediate tables we derived from the three tables all have primary keys as part of the attributes, and all the join operations involve primary keys, the last intermediate table would not contain duplicates. Thus the projection operation costs 500 * 2 = 1000 I/O costs.

The total I/O cost is 3000 + 4000 + 19389 + 56556 + 1000 = 83945 block reads and writes

2.





Implementation:
In this organization we will still use the same query tree but change some of the evaluation methods. We still use one loop over each table for projection since all projections involve primary keys. Selection on P still costs linear time. The join between E and D is pushed lower than the join of P and D, since the sort-merge method can be used in the former join.

Costs:
At the leaves level projections on D, E and P cost is (B_D + B_E + B_P) * 2 = 7000 block reads and writes.

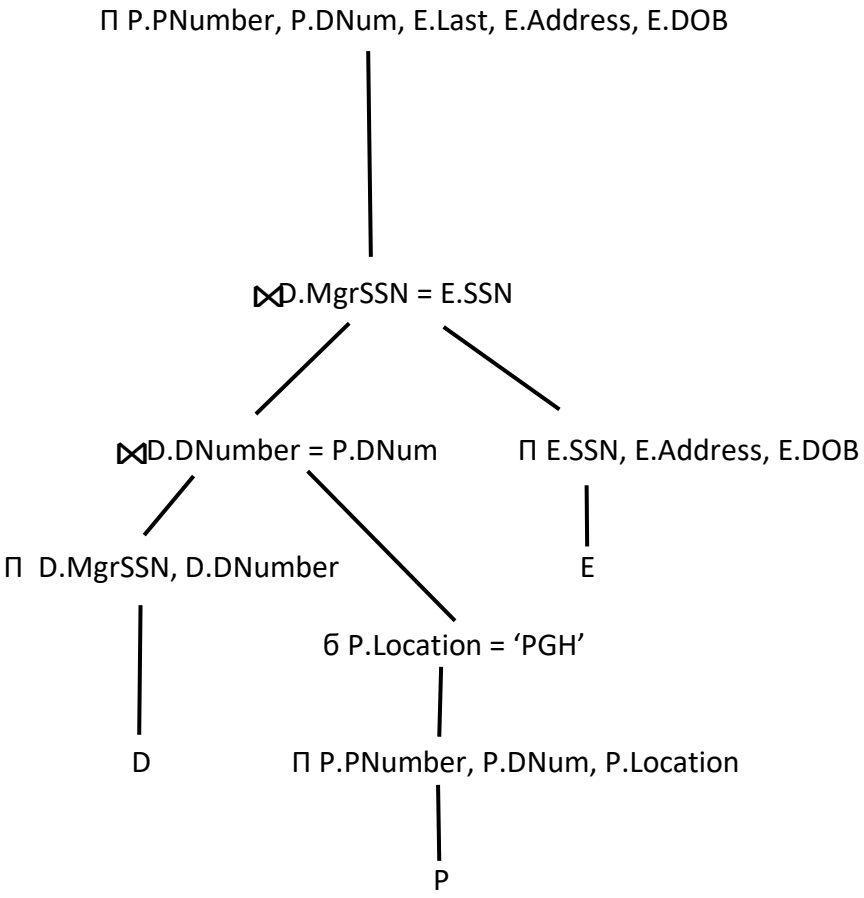
In the next level we can perform sort-merge join on D and E. Since D and E relations are already sorted, we can simply scan both relations and bring pairs of blocks into memory (assuming both relations are sorted by Dept. Number), then write the joined blocks back to disk. Thus, the level I/O cost is B_D + B_E + B_D = 500 + 1000 + 500 = 2000. The resulting cardinality is B_D since we are joining by E's primary key. At the same level we can perform selection on P. This costs a linear scan on P. After string comparison, we write B_P / 2 = 500 back to disks. The second half of the process costs 500 + 2000 = 2500. In total, this level costs 2000 + 500 = 2500 block reads and writes.

At this level the intermediate tables are no longer sorted so we use the nested loop join. The cost would be 500 + (500 / (20 - 2)) * 500 = 14389 block reads and writes. We will write 500 blocks back to disk so the total I/O cost at this level is 14389 + 500 = 19389

At last, the projection operation costs scanning and writing all blocks = 2* 500 = 1000 since the last intermediate table contains no duplicates.

The total I/O cost is 7000 + 2500 + 19389 + 1000 = 29889

3.



Implementation:
In this organization we can use the same query tree in the first section. Since all the three relations are in heap files, projections and selection operations have the same costs as before. Only P has an access method, we want to do a join on P as soon as possible to reduce the subsequent I/O costs. When performing the join on D and P, we can single-loop join method, iterating through the tuples in D and using the hashing access method to bring in matching records from P. D's cardinality is less than E so we prefer to have P join with D first.

Costs:
At the first level we do projections on D and P, reading and writing all blocks costing (500 + 1000) * 2 = 3000.

Next we perform selection on P, causing 1000 block reads and 500 writes (as described in section 1). We also perform projection on E at this level, causing 2000 block reads and writes. The I/O cost in total is 1000 + 500 + 2000 * 2 = 5500

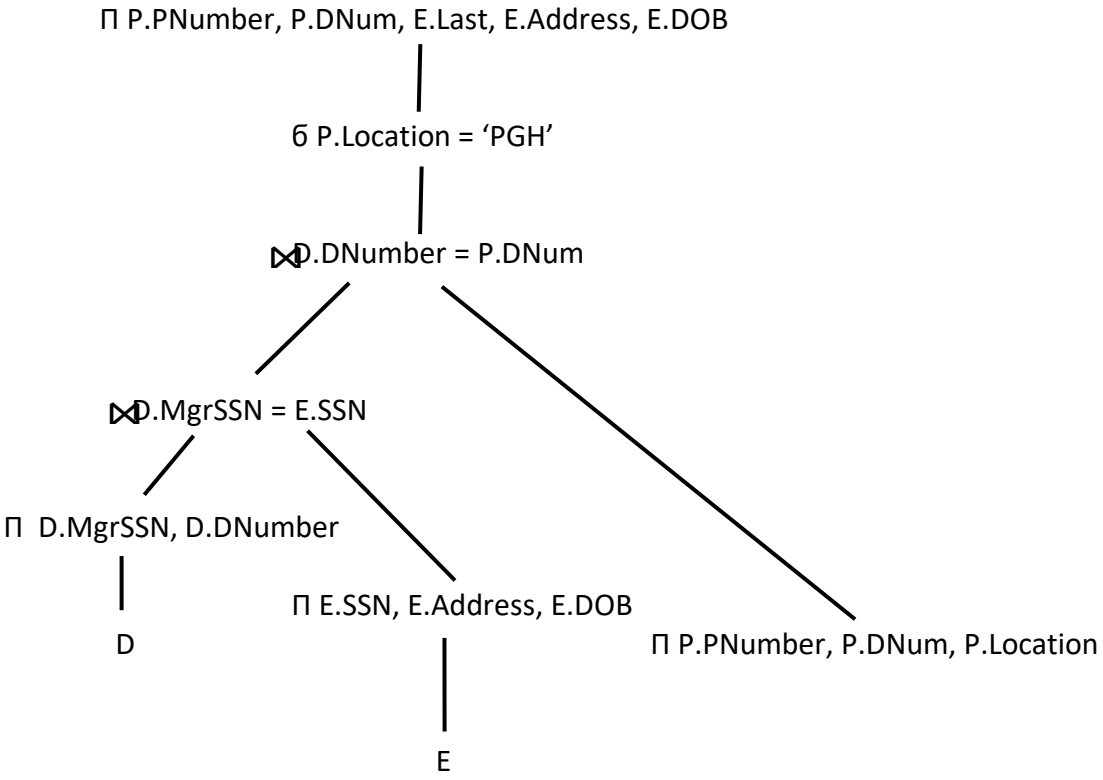
Then we perform single-loop join operation on D and P. We set the the relation with access method, P in inner loop and the other relation, which is D, as the outer loop. We bring every block of D to the memory and for each record in P's blocks, we bring in a block with the

matching record from P, which costs R_D * 1 with hashing access method. We now have 500 blocks to write to disk. This results in a total cost of 500 + 2500 + 500 = 3500.

And then we perform a nested-loop join on the last two intermediate tables, setting the one derived from D and P as the outer loop. This operation costs 500 + (500 / (20 - 2)) * 2000 = 56056 (as explained before). We have 500 blocks to write here (join by E's primary). Thus, this level is 56056 + 500 = 56556.

In total, this page organization's I/O cost is 3000 + 5500 + 3500 + 56556 = 68556

4.



Implementation:
In this organization, E is a sorted file and also the largest file. We want to take advantage of this fact and join E and D at the earliest stage to reduce further overhead. And luckily, an access method exists in relation D so we can perform single-loop join on these two relations. We don't want to join D and E first, since access methods exists in both relations and we want to use their access methods in two joins instead of one. However, we want to perform the selection operation after we perform join this time since the access method could not be used on the intermediate table derived from P. Also, due to the fact that D has less tuples than P we assign the index access method to D and hash access method to P, whereby a lesser total I/O cost will be incurred compared to the other way around.

Costs:
At the leaves level, we perform projection on D and E. As described before this operation costs (B_D + B_E) * 2 = (500 + 2000) * 2 = 5000 with linear scan method to read blocks and write intermediate table back to disk.

At the next level, we perform the equi-join of D and E. We use single-loop join and scan E and access method to fetch matching records from D. When finding the matching records of every E's record, we perform index searching on D and bring back the containing blocks. We assume that D has B+ tree index and the degree is 6. Thus, the cost of each record access is $\log_6 500 = 4$ (levels) and the cost of this join is B_E + (R_E * (L_D + 1)) = 2000 + (10000 * (4 +1)) = 12000. The result takes 500 blocks to write or 2500 records (join by E's primary keys). Then we perform projection on P with a linear search that costs 1000 *2 block reads and writes. The total cost in this level is 12000 + 500 + 2000 * 2= 16500

Now, with 2500 records in this intermediate table, we join it with the projected P. Using single-loop join operation, we scan the intermediate table produced by D and E join, and then fetch the matching records from P, costing B_D + (R_D * 1) = 500 + 2500 = 3000. We have 1000 blocks to write here (join by D's primary key). Thus the total cost becomes 3000 + 1000 = 4000

Then, we performed selection on this intermediate table, which takes a linear search, costing 1000 and writes 500 (deduced by the string). Total I/O cost is 1500.

Finally, we perform an projection with the intended attributes. This step costs 500 as explained.

The total I/O cost is 5000 + 16500 + 4000 + 1500 + 500 = 27500