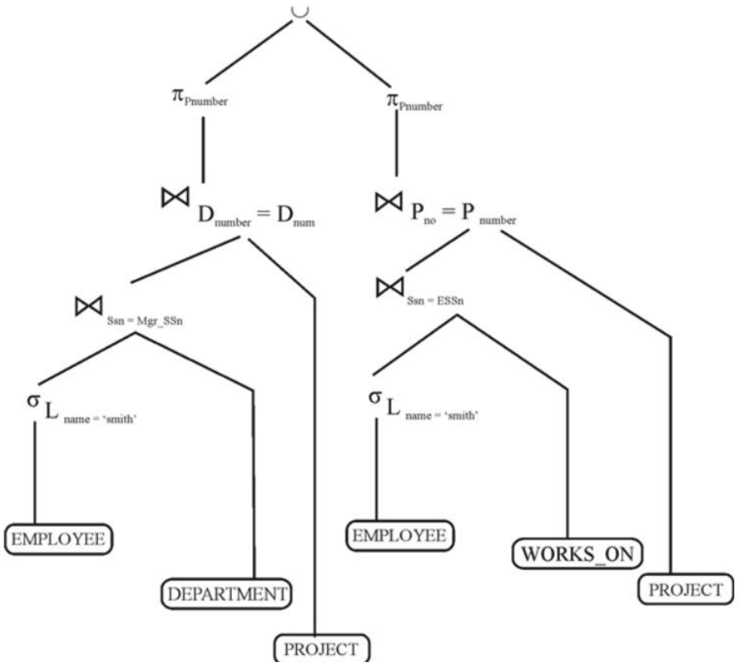
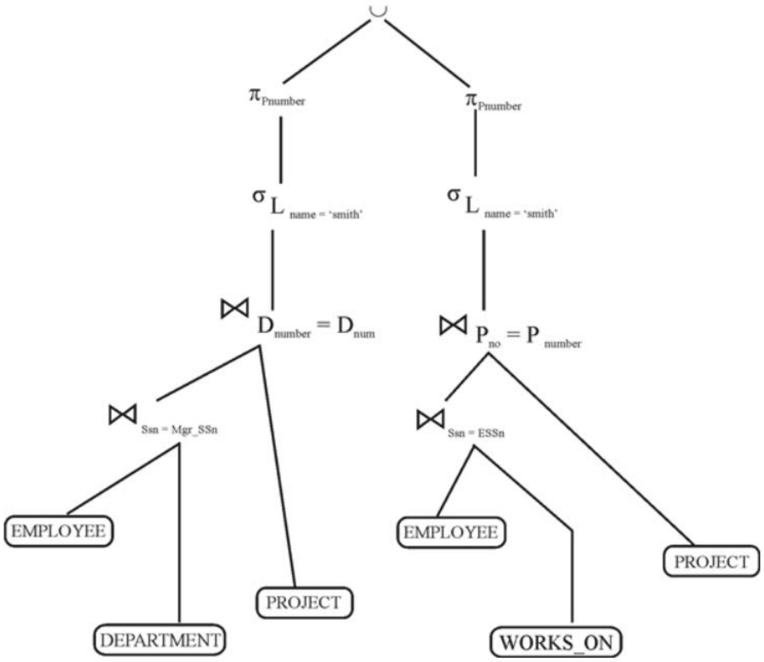
18.13)

a)



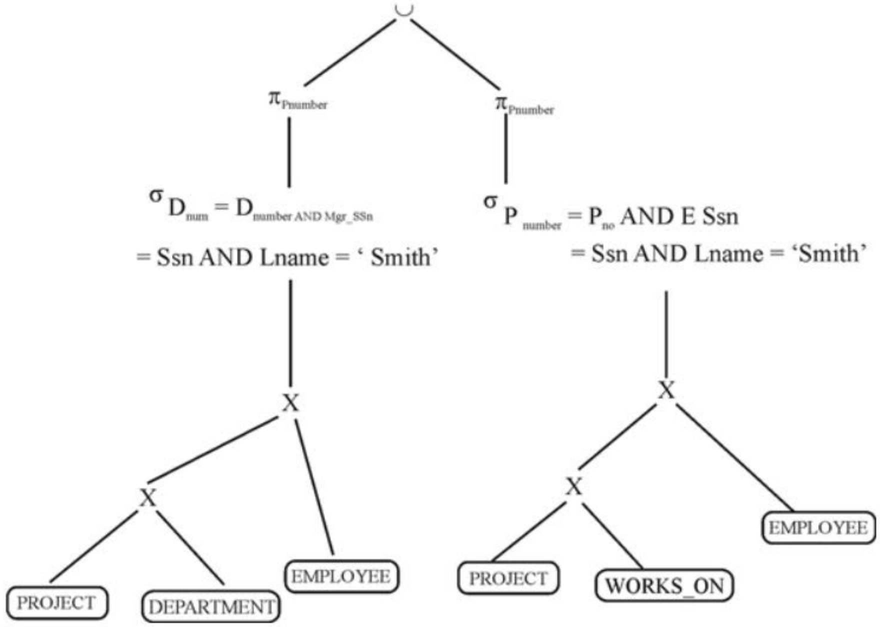
This tree would be good when the number of attributes is small, although when projecting before joining, this may cause more overhead and larger execution times.



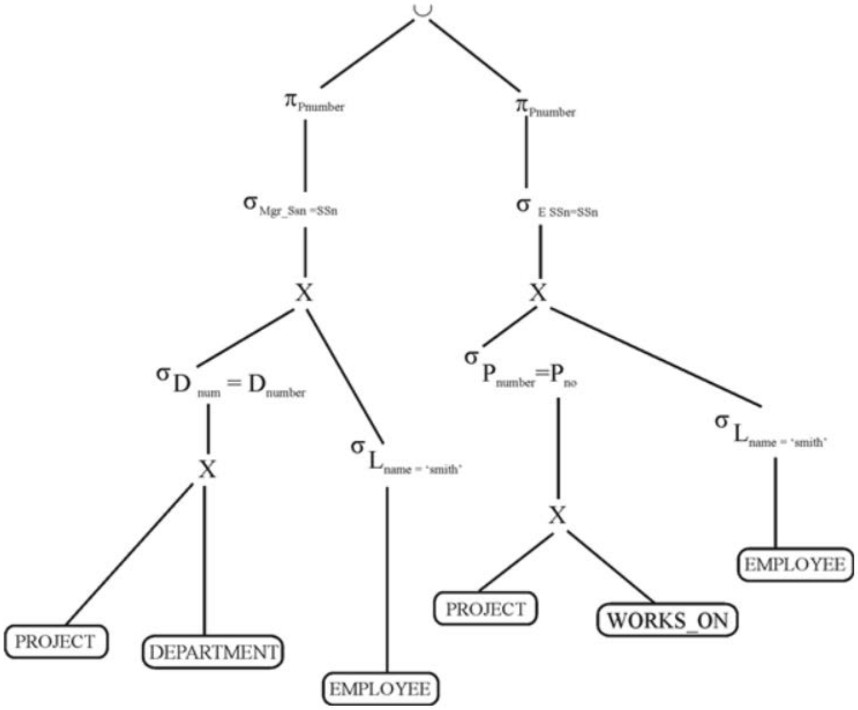
This tree would be ideal if the number of employees is small because the select is before the join so the initial query for tuples would be fairly large.

b)

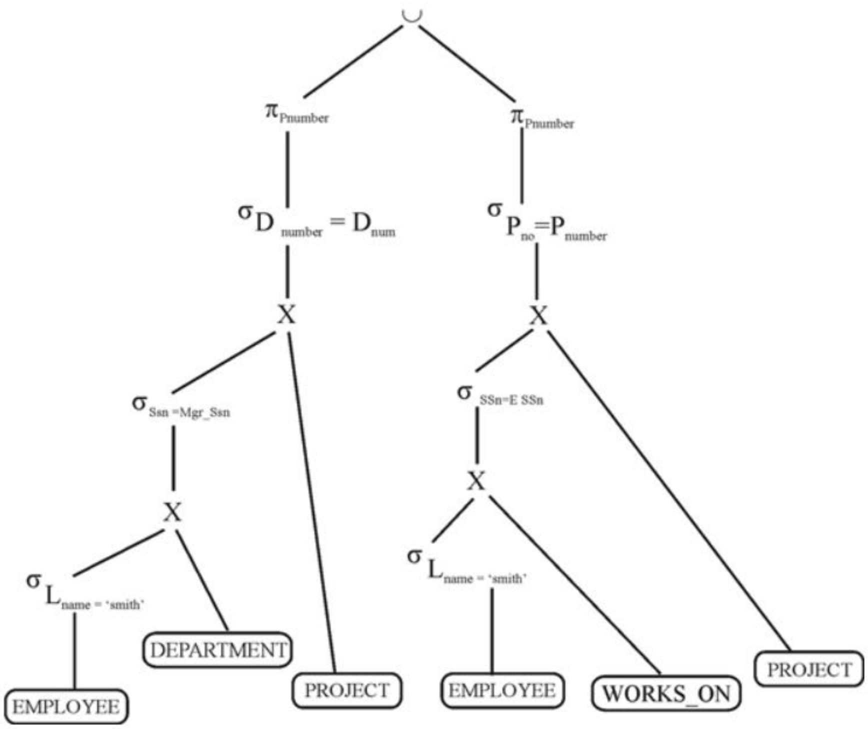
Initial Tree:



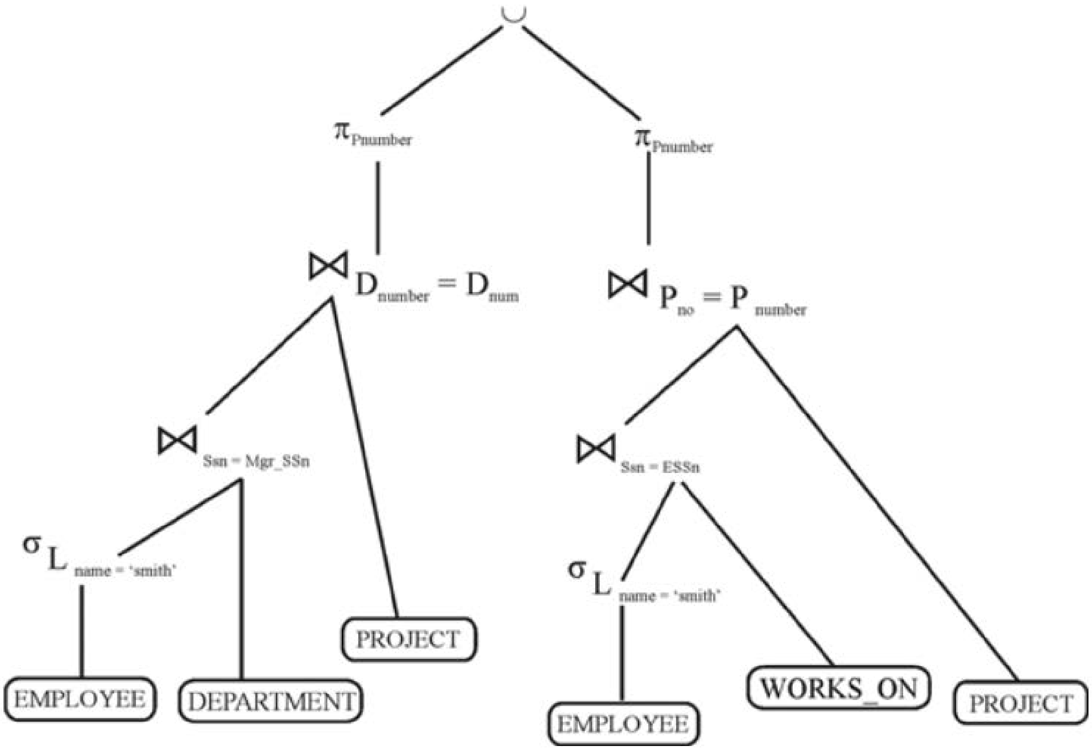
Step 1: SELECT operations are moved down.



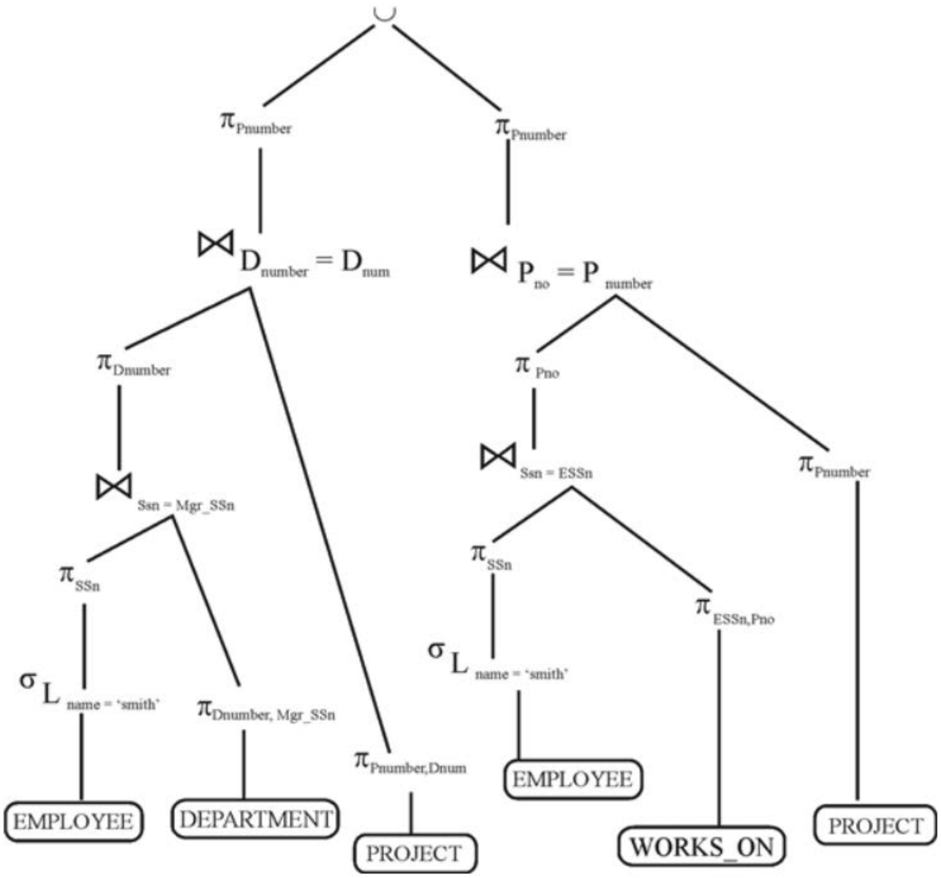
Step 2: Apply SELECT operation first.



Step 3: Replace CARTESIAN and SELECT with JOIN



Step 4: Project operations are moved down, this is the final tree:



c)

The first tree in question A, can be optimized further by moving the PROJECT operations down. The second tree in A is also not the final tree, it can be optimized further by moving the SELECT operation down the tree.

19.15)

Assume relations R and S which are stored in bR and bS disk blocks and the result is stored in bRESULT.

For PROJECT operation:

If PROJECT includes a key of R, then the cost is 2*bR, since read and write have the same size which is R. If PROJECT does not include a key of R then, the mid-results must be sorted before eliminating the duplicates which means a second read must be used, so the cost will be 3*bR +k*bR*log2bR.

For DIFFERENCE and INTERSECTION:

Sort, scan/merge

k*[(bR*log2bR) + (bS*log2bS)] + bR + bS + bRESULT

For Cartesian Product:

Assume two memory buffers and nB buffers are used.

C R X S = bR + ceiling(bR/nB -1))*bS)+(|R|*|S|)bfr RS

19.18)

Nested loop approach:

The number of buffer's available in the main memory for join is nB, so nB=7 blocks

The DEPARTMENT file consists of r_b=50 records

Stored in b_D = 10 disk blocks.

The Employee block consists of r_E = 6000 Records stored in b_E = 2000 disk blocks.

(nB -2) blocks of the employee file is read so the total number of block access for the outer file = bE.

Number of (nB -2) block of the outer file is loaded = [b_E/(nB-2)]

Combining all these the two outer block and the inner block, we get the following number of block accesses:

B_E +([b_E/(nB-2)*b_D])

=2000 + ([(2000/5)]*10)

=6000 block accesses

If we reverse the loop order then we have the following number of block access:

10+([10/5])*2000

=4010 block accesses.

2) Joining r1, r2 and r3 will be the same no matter in which way they are joined due to the associative and commutative properties of join. So we will consider the size based joining r1 and r2 and then r3 (r1 $\bowtie$ r2) $\bowtie$ r3. Joining R1 and R2 will give at most 1000 tuples since the key is C (common between R1 and R2), next joining with R3 “E” will then be the key, thus once again the final relation will have at most 1000 tuples. An efficient strategy would be to use a nested loop algorithm where R1 would be in the outer loop and then we set an index for attribute C in R1, then we look for a match in R2 with the index on C which will match one tuple. Similarly once R1 and R2 are joined, we now set an index on attribute E and look for a match in R3 to join with.

3) R1 will need 800 blocks (20000/25) and R2 will need 1500 Blocks (45000/30). If we have M pages of memory and if M>900, the join can be done in 1500+800 disk access, if M<=800 then we can look at a nested loop join as below.

For a nested loop join, we can either have r1 on the outer loop or r2 on the outer loop. If r1 is the outer relation then we will have 20000*(45000/30)+(20000/25) = 30,000,800 disk accesses. If R2 is the outer relation then we need 45000*(20000/25)+(45000/30) = 36,001,500 disk accesses, thus having r1 as the outer relation is the more efficient way of doing it.

4) A way to think about this is to first determine the average number of tuples which would be joined with each tuples of the second relation. For example, for each tuple in r1, 1500/V (C, r2) = 15/11 tuples (on the average) of r2 would join with it, so the intermediate relation will have 15000/11 tuples. We then join it with r3 which will have about (15000/11 x 750/100) = 10227 tuples. An efficient strategy to compute join would be to join r1 and r2 first since the intermediate relation will be roughly the same size and then join it with r3 to get the final relation.