CSE 444 – Homework 5
Query Optimization

Name: Anupam Gupta

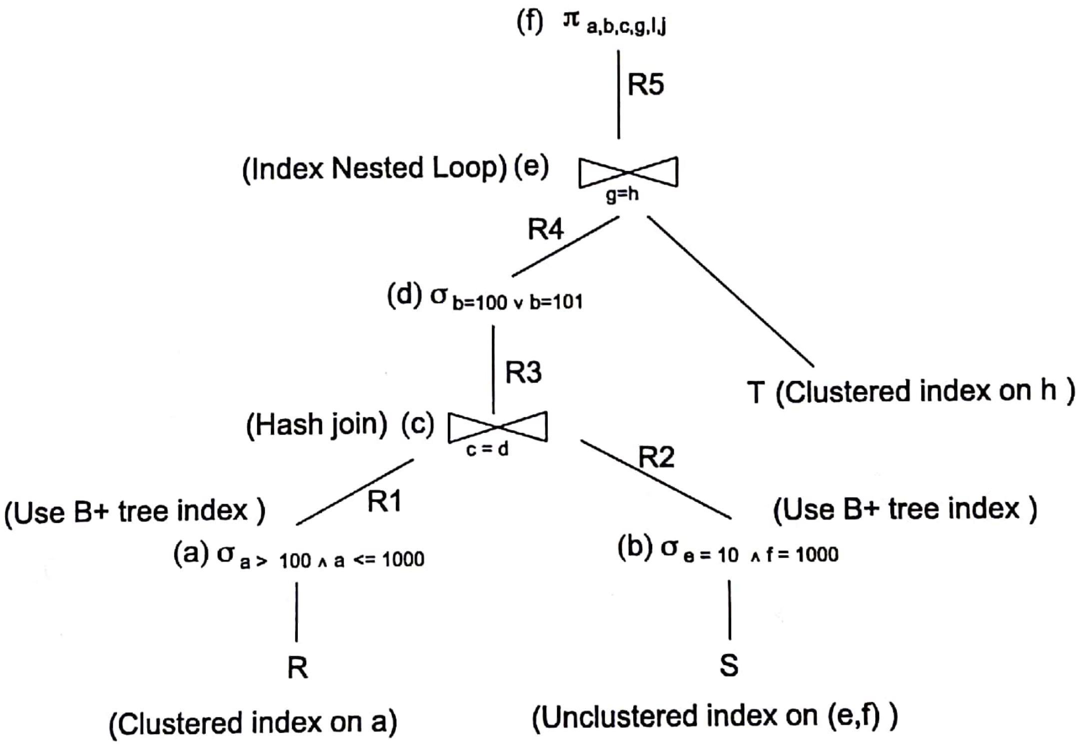
Question	Points	Score
1	20	
2	20	
Total:	40	

1 Query Plan Cost Computation

1. (20 points)

   Consider the following relations and physical query plan:

R(a,b,c)	S(d,e,f,g)	T(h,i,j)
B(R) = 1000	B(S) = 1000	B(T) = 1000
T(R) = 10,000	T(S) = 10,000	T(T) = 10,000
Min(R,a) = 0	V(S,e) = 10	V(T,h) = 100
Max(R,a) = 9000	V(S,f) = 100	
V(R,b) = 100	V(S,d) = 50	
V(R,c) = 20	V(S,g) = 40	



(a) (7 points) Compute the selectivity of the following predicates:

1. $a > 100 \wedge a \leq 1000$
2. $e = 10 \wedge f = 1000$
3. Join predicate: $c = d$
4. $b = 100 \vee b = 101$
5. $g = h$

1.) $a > 100$ & $a \leq 1000$ $= \dfrac{900 \text{ range}}{\max(R,a) - \min(R,a)} = \dfrac{900}{9000} = \dfrac{1}{10}$

2.) $e = 10 \wedge f = 1000$: Selectivity of $e=10$ * Selectivity of $f = 1000$

$= \dfrac{1}{10}\left(= \dfrac{1}{V(S,e)}\right) * \dfrac{1}{100}\left(= \dfrac{1}{V(S,f)}\right)$

$= \dfrac{1}{1000}$

3.) $c = d$ $\Rightarrow \dfrac{1}{\max(V(R,c), V(S,d))} = \dfrac{1}{\max(20,50)} = \dfrac{1}{50}$

4.) $b = 100 \vee b = 101$ = (Selectivity of $b=100$) + (Selectivity of $b = 101$)

$= \dfrac{1}{100}\left(= \dfrac{1}{V(R,b)}\right) + \dfrac{1}{100}\left(= \dfrac{1}{V(R,b)}\right)$

$= \dfrac{2}{100} = \dfrac{1}{50}$

5) $g = h$ $= \dfrac{1}{\max(V(S,g), V(T,h))} = \dfrac{1}{\max(40,100)} = \dfrac{1}{100}$

(b) (7 points) Compute the cardinality of all intermediate relations labeled R1 through R5 and the final result, call it R6.

$R1 \Rightarrow \underbrace{\frac{1}{10}}_{\text{selectivity}} * T(R) = \frac{1}{10} * 10000 = 1000$

$R2 = \underbrace{\frac{1}{1000}}_{\text{selectivity}} * T(S) = \frac{1}{1000} * 10000 = 10$

$R3 = \text{Selectivity} * \text{Cardinality}(R_1) * \text{Cardinality}(R2)$

$= \frac{1}{50} * 1,000 * 10 = 200$

$R_4 = \text{Selectivity} * \text{Cardinality}(R3)$

$= \frac{1}{50} * 200 = 4$

$R_5 = \text{Cardinality}(R_4) * T(T) * \text{Selectivity}$

$= 4 * 10,000 * \frac{1}{100} = 400$

$R_6 = R_5$ = Since just projection = 400

(c) (6 points) Compute the cost of this query plan in terms of number of pages read from disk or written to disk. Assume that all index pages are in memory at any time. Also assume that the hash table for the hash join will fit in memory.

To read Cardinality(R_1) = 1000 tuples from disk we will need $\frac{1000}{\frac{T(R)}{B(R)}}$ page reads from disk since there is a clustered index on a already. Hence, no. of page reads = $\frac{1000}{\frac{10,000}{1,000}}$ = 100 (for R_1)

Since there is an unclustered index on (e, f) on S already, to read cardinality (R_2) = 10 tuples, we would need 10 page reads from disk since the index is unclustered.

R_3 and R_4 operations are on the fly since the pages of tuples have already been read from the disk

For R_5, we need to read T from disk. We need to read Cardinality (R_5) =400 tuples from T. Since, there is a clustered index on h already, we need to do $\frac{400}{\frac{T(T)}{B(T)}}$ page reads from disk = $\frac{400}{10}$ = 40 I/O reads.

∴ Total I/O cost of query = 100 + 10 + 40 = $\boxed{150}$

2 Query Optimization

2. (20 points)

Consider the following three relations:

R(a,b,c)	S(d,e)	W(f,g,h)
B(R) = 100	B(S) = 1000	B(W) = 10
T(R) = 1,000	T(S) = 10,000	T(W) = 100

Consider the following SQL Query:

```
SELECT *
FROM R, S, W
WHERE R.a = S.d
AND   R.c = W.h
```

(a) (10 points) Assume that all relations are stored in heap files, there are no indexes, only page-at-a-time nested-loop joins can be used, and the selectivity of each join predicate is 0.1%.

Show the query plan selected by a Selinger-style, bottom-up, dynamic programming optimizer. Use the number of disk IO operations as the cost function.

Hints:

- Remember that a Selinger-style optimizer will try to avoid Cartesian products in the plan. So do NOT consider such plans.
- The Selinger optimizer will only consider left-deep plans.

Draw the selected plan and show how it is derived. You can use the following table to help you but do NOT worry about computing the exact cost and size values if you don't need exact values to prune plans. In the table, P/K indicates the choice to either prune or keep the subplan. Hint: When joining tuples, keep in mind that the tuples get bigger.

Subquery	Cost	Size of output	Plan	P/K
R	100 page IOs	1K records on 100 pages	Sequential scan of R	K
S	1K page IOs	10K records on 1K pages	Sequential scan of S	K
W	10 page IOs	100 records on 10 pages	Sequential scan of W	K
RS	...	10K records on 2K pages	...	
SR	...		...	
...				

Question 2. a

Sub Query	Cost	Size of Output	Plan	P/K
R	100 Page I/Os	1K records in 100 pages	Sequential Scan of R	K
S	1K Page I/Os	10K records in 1K pages	Sequential Scan of S	K
W	10 Page I/Os	100 records in 10 pages	Sequential Scan of W	K
RS	100.1K Page I/Os	10K records in 2K pages	R as outer relation join (S)	K
SR	101K Page I/Os	10K records in 2K pages	S as outer relation join (R)	P
RW	1.1K Page I/Os	100 records in 20 pages	R as outer relation join(W)	P
WR	1.01K Page I/Os	100 records in 20 pages	W as outer relation join(R)	K
(RS)W	20K Page I/Os (final calculation shown below table = 120.1K)	1000 records in 334 pages	RS as outer relation join(W)	P
(WR)S	20K Page I/Os (final calculation shown below table = 21.01K)	1000 records in 334 pages	WR as outer relation join(S)	K

The total Cost for (RS)W = Cost of joining RS and W + Cost of joining RS = 20K + 100.1K = 120.1K Page I/Os

Total Cost for (WR)S = Cost of joining WR and S + Cost of joining WR = 20K + 1.01K = 21.01K Page I/Os

Some intermediate plans are not considered either since they were not left deep plans or they were Cartesian products. Previously pruned subqueries' extensions are not shown since those plans were already pruned and not considered further. For example, (SR)W and (RW)S in this case are not considered since SR and RW had already been pruned before. Hence, we won't go ahead with their plans.

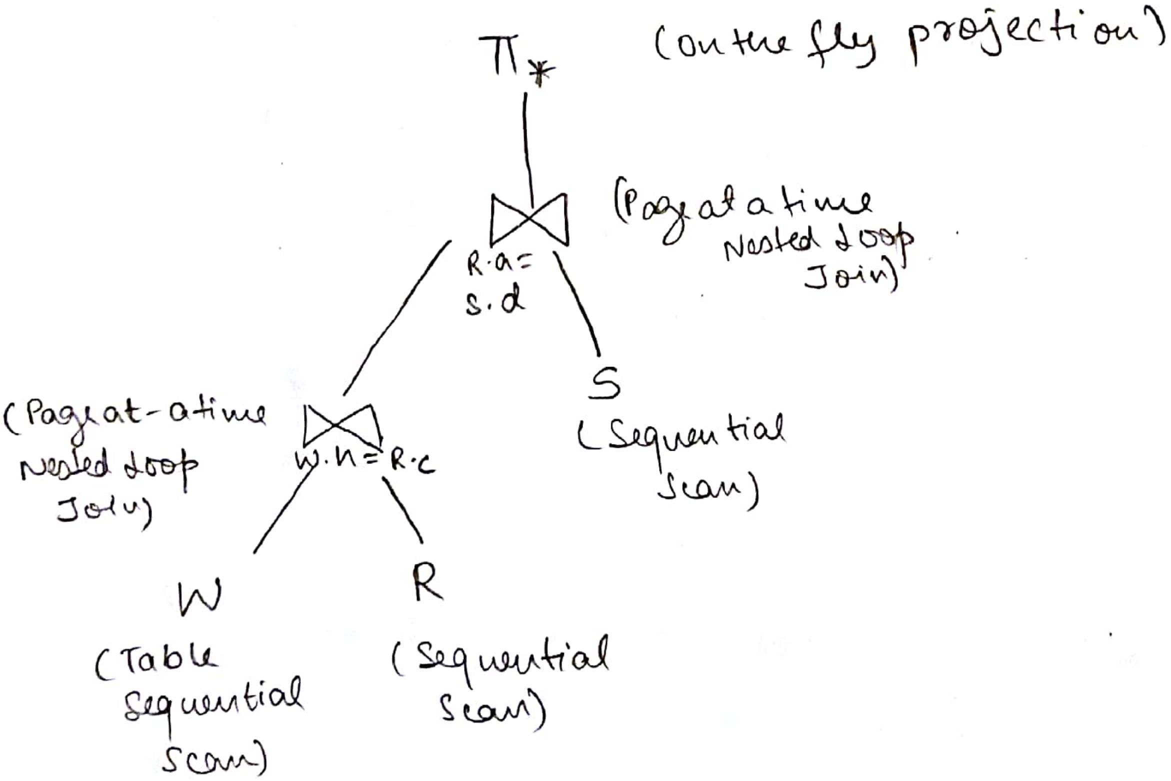
More space for your answer:

As shown in the table the best plan is:

i) Join W and R first (W being the outer relation)

ii) Join WR with S (WR being the outer relation)

Hence, the best plan is:



(b) (5 points) Consider the following small modification to the above query and consider that we add an unclustered B+ tree index on S.d as well as a clustered B+ tree index on R.b. Without re-computing the new best plan, explain how these changes affect the optimization process for this query.

```
SELECT *
FROM R, S, W
WHERE R.a = S.d
AND   R.c = W.h
AND   R.b > 100
```

If we have an unclustered B+ tree index on S.d and a clustered B+ tree index on R.b, we can reduce the cost of reading S since we no longer need to do a sequential scan of S. We can use the index on S.d to get just the values that match. In addition, for R, we also don't need to do a full table scan since we have a clustered index on R.b, we can keep reading R till R.b>100 in contiguous memory slots (since it is clustered). This reduces the cost of reading R from the disk and effectively the cost of joining R and S while R.b >100. This changes the optimization of the query since the overall cost of some joins change considerably which makes plans that include these joins also better. Due to the additional indices, some of the joins that previously had higher cost (especially b/w R & S) will have a lower cost and might result in a better (lower cost) final optimized plan

(c) (5 points) What is an interesting order? Give one or two examples and explain how they can be useful in getting a better physical query plan.

An interesting order is an order that resultant tuples are arranged in that might come in handy further up in the query and help find a better, lower cost plan. When subplans are pruned, plans with interesting orders (even if they have a higher cost) are not pruned in the Sellinger optimizing algorithm.

For example. suppose we have this SQL Query.

```
SELECT *
FROM R, S, T
WHERE R.a = S.d
 AND T.b = S.c
```

Suppose S is indexed on both c (clustered) and d (unclustered). While joining R and S, it might seem better to use the 'd' index of S since R.a = S.d. In that case we would prune the subplan where R.a = S.d but we use the 'c' index of S. However, since the order of the elements we receive in one index is different than the one with the other index, we don't prune. In addition, for this query, depending on the sizes of R, S and T, using and retaining the results from the join using the 'c' index might actually result in a lower cost plan when joining with T. Hence, interesting order retainment can be useful in getting a better physical plan in this case.