Problem Set 3: Query Evaluation and Optimization

COMP_SCI 339

Due: Monday, November 6, 2023

Name: Zhuoyuan Li

1 Buffer Manager and Simple Query Execution

1. (3 points) Consider a buffer pool with a LRU page replacement policy. It has 10 pages. Each page has 8 KB (8192 bytes) and we support 32-bit integers. The buffer pool is initially empty.

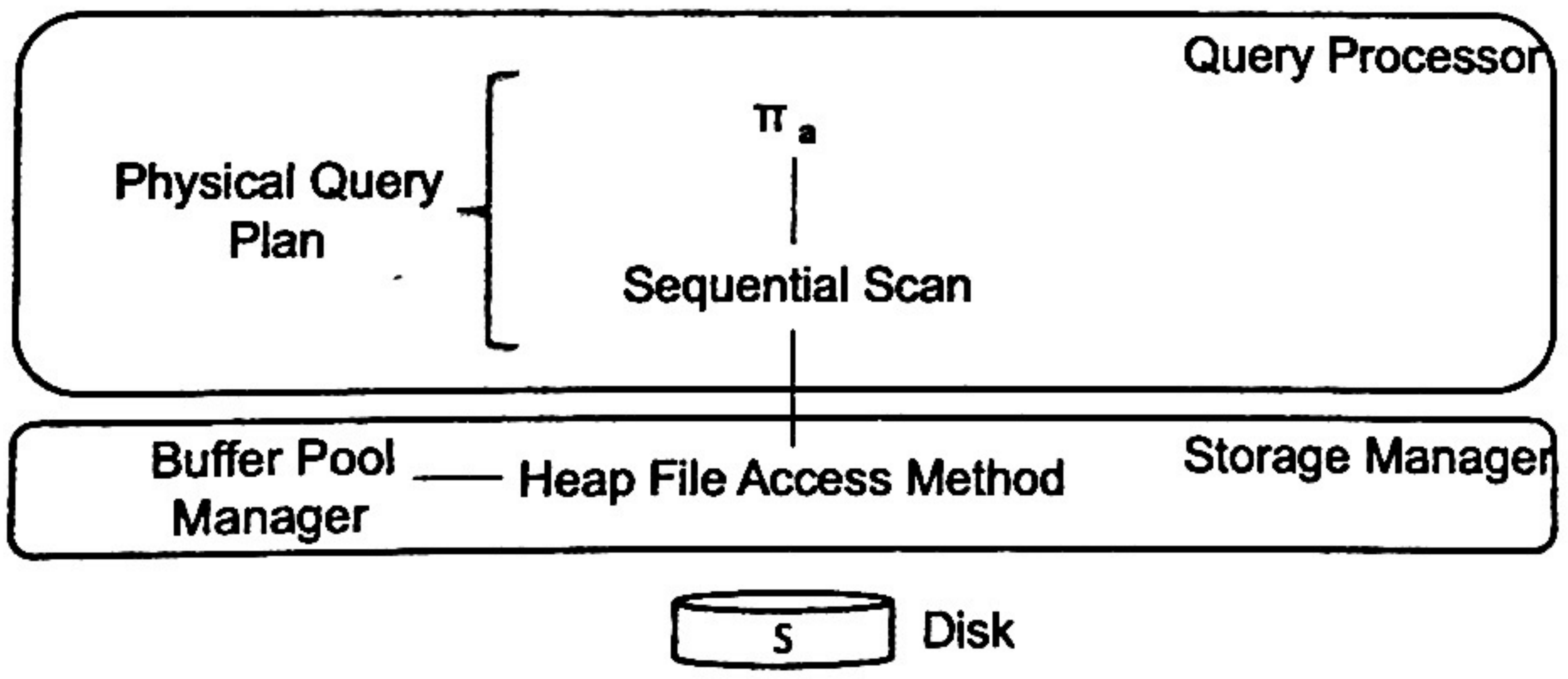
   Recall the relation S – from Problem Set 2 – with the following schema:

   `S(a int, b char(10), c char(20))`

   and the following SQL query:

   ```
   SELECT S.a
   FROM S
   ```

   A simple *physical query plan* for this query is the following:



   We show the query plan in the context of the relevant DBMS architecture components. To execute the query, the system will call `open()` and then `next()` on the project operator. We ignore `hasNext()` in this exercise.

   Consider that relation S is stored in a heap file on disk.

(a) (1 point) Explain how the execution of this query will proceed as the system calls `open()` and then `next()` on the topmost, project operator. You only need to describe what happens on the call to `open()` and then on the first call to `next()`. You do not need to describe subsequent calls to `next()`.

Your explanation should describe the control flow (who calls whom and when) between (1) the project operator, (2) the sequential scan operator, (3) the heap file access method, and (4) the buffer pool manager. Keep in mind that the buffer pool manager will call the heap file to actually read a page from disk.

when open() is called, the Project operator is invoked first, then, the sequential Scan operator is triggered. The sequential scan operator then calls the heap file access method to retrieve page from the buffer pool. Since buffer pool initially is empty, it then invokes the buffer pool manager to load one page to the buffer pool.

when next() is called, the Project operator triggers the sequential operator. Then, the sequential operator requests the heap file access method to load the page from buffer pool. Since the buffer pool manager has provided page in open(), it passes the page to the sequential operator so it can read the first tuple data. Finally, the read tuple returned to the project operator, which returned the projected result S.a.

(b) (0.5 point) What will page(s) will be in the buffer pool after the first **next()** call on the project operator returns a tuple.

there will be one page containing tuples from S in the buffer pool after the first next() has been called.

(c) (0.5 point) What will be the contents of the buffer pool after the second **next()** call on the project operator returns a tuple.

It will have the same page as the fist next() Call

(d) (0.5 point) Assuming S contains 10,000 tuples, what will be the content of the buffer pool after 1000 next() calls on the project operator.

Based on relation S, each tuple has size 34 bytes, and each page can hdd up to 240 tuples. Therefore, if there are 1000 next() calls, we need $\lceil \frac{1000}{240} \rceil = 5$ pages from buffer pool. Thus, now buffer pool should have 5 pages loaded.

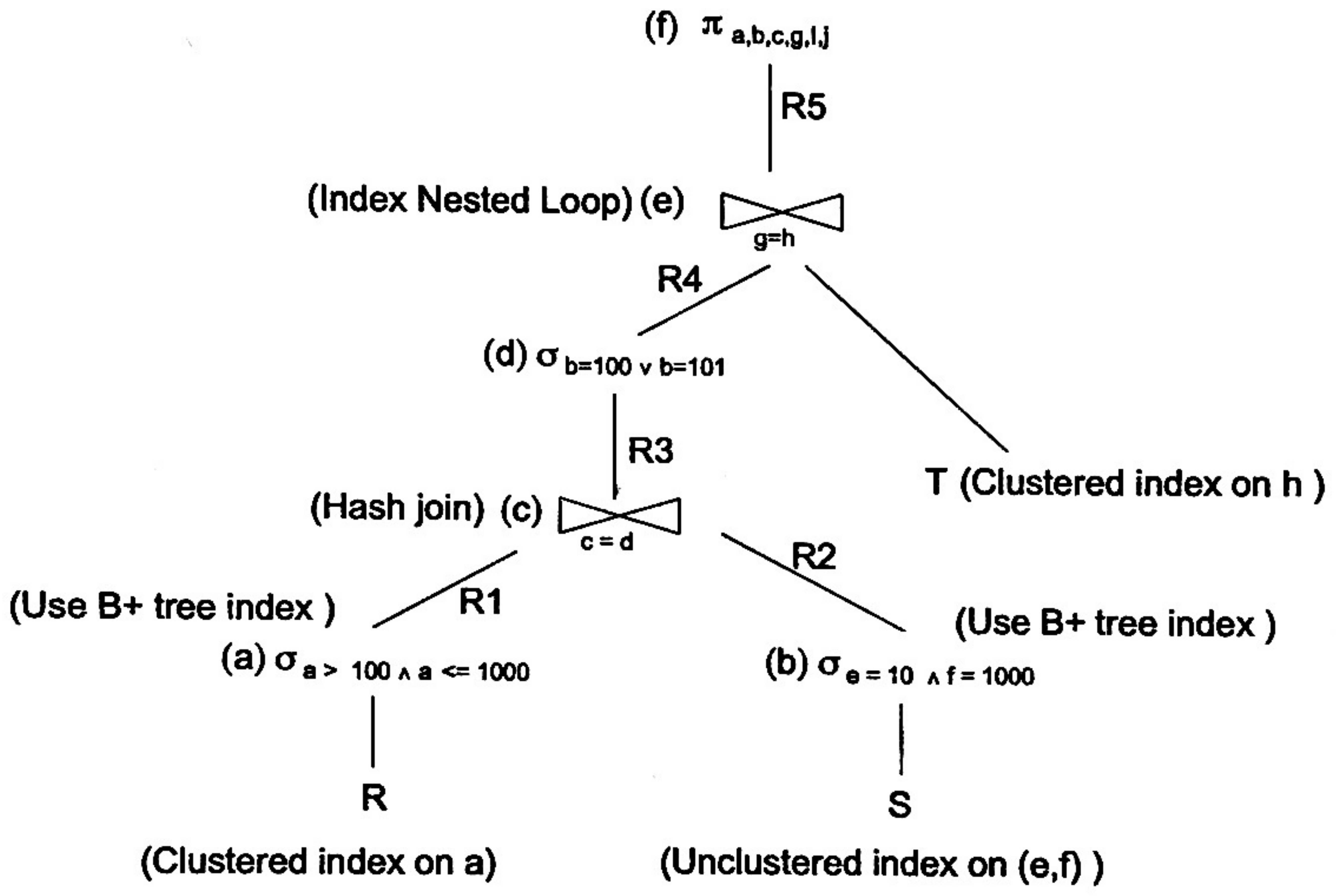
(e) (0.5 point) Assuming S contains 10,000 tuples, what will be the content of the buffer pool after the query completes?

Since we need to finish running the query here, we need 10000 calls for next(). Therefore, we need at least $\lceil \frac{10000}{240} \rceil = 42$ pages from the buffer pool. However, it only has 10 pages space. Since we are using LRU page replacement policy, the least recently used page will be replaced by the most recently requested page. Thus, the result buffer pool will be consisted of page 33 - page 42.

2 Query Plan Cost Estimation

2. (4 pts) Consider the following relations and physical query plan:

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



(f) $\pi_{a,b,c,g,I,j}$
R5
(Index Nested Loop) (e)
g=h
R4
(d) $\sigma_{b=100 \vee b=101}$
R3
T (Clustered index on h)
(Hash join) (c)
c = d
R2
R1
(Use B+ tree index)
(Use B+ tree index)
(a) $\sigma_{a > 100 \wedge a <= 1000}$
(b) $\sigma_{e = 10 \wedge f = 1000}$
R
S
(Clustered index on a)
(Unclustered index on (e,f))

(a) (1 point) Compute the selectivity of the following predicates:

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

5. $\text{Selectivity}(g = h) = \dfrac{1}{\text{MAX}(\text{ICARD}(S,g), \text{ICARD}(T,h))} = \dfrac{1}{\text{MAX}(40,100)} = \dfrac{1}{100}$

1. $\text{Selectivity}(a > 100 \wedge a \leq 1000) = \dfrac{100 < a \leq 1000}{\text{MAX}(R,a) - \text{Min}(R,a)} = \dfrac{900}{9000} = \dfrac{1}{10}$

2. $\text{Selectivity}(e = 10 \wedge f = 1000) = \dfrac{e = 10 \wedge f = 1000}{\text{ICARD}(S,e) \cdot \text{ICARD}(S,f)} = \dfrac{1}{1000}$

3. $\text{Selectivity}(c = d) = \dfrac{1}{\text{MAX}(\text{ICARD}(R,c), \text{ICARD}(S,d))} = \dfrac{1}{\text{MAX}(20,50)} = \dfrac{1}{50}$

4. $\text{Selectivity}(b = 100 \vee b = 101) = \dfrac{1}{\text{ICARD}(R,b)} + \dfrac{1}{\text{ICARD}(R,b)} = \dfrac{1}{100} + \dfrac{1}{100} = \dfrac{1}{50}$

(b) (1.5 points) Compute the cardinality of all intermediate relations labeled R1 through R5 and the final result, call it R6.

$|R1| = \text{NCARD}(R) \cdot \text{Selectivity}(a > 100 \wedge a \leq 1000) = \dfrac{10000}{10} = 1000$

$|R2| = \text{NCARD}(S) \cdot \text{Selectivity}(e = 10 \wedge f = 1000) = \dfrac{10000}{1000} = 10$

$|R3| = |R1| \cdot |R2| \cdot \text{Selectivity}(c = d) = \dfrac{1000 \cdot 10}{50} = 200$

$|R4| = |R3| \cdot \text{Selectivity}(b = 100 \wedge b = 101) = \dfrac{200}{50} = 4$

$|R5| = |R4| \cdot \text{NCARD}(T) \cdot \text{Selectivity}(g = h) = \dfrac{4 \cdot 10000}{100} = 400$

$|R6| = |R5| = 400$

(c) (1.5 points) Compute the cost of this query plan in terms of number of pages read from disk or written to disk. Assume that all of the operators after the hash join are pipelined - i.e., you do not need to store intermediate results on disk. Only count data pages, not index pages, in your lookups.

Cost for R1 : $\frac{1000}{10} = 100$ I/Os

Cost for R2 : $\frac{10000}{1000} = 10$ I/Os

Cost for R3 : $(C1 + C2) \cdot 3 = (100 + 10) \cdot 3 = 330$ I/Os

Cost for R4 : 0 since there is no more pages need to be loaded

Cost for R5 : $\frac{|R4| \cdot TCARD(T)}{ICARD(T,h)} = \frac{4 \cdot 1000}{100} = 40$ I/Os

Cost for R6 : 0 Since there is no more pages need to be loaded.

Therefore, the total cost of this query plan will be:

$100 + 10 + 330 + 0 + 40 + 0 = 480$ I/Os

3 Query Optimization

3. (4 points)

Consider the following three relations:

R(a,b,c)	S(d,e)	W(f,g,h)
TCARD(R) = 100	TCARD(S) = 1,000	TCARD(W) = 10
NCARD(R) = 1,000	NCARD(S) = 10,000	NCARD(W) = 100

Consider the following SQL Query:

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

(a) (2 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 I/O 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 I/Os	1K records on 100 pages	Sequential scan of R	K
S	1K page I/Os	10K records on 1K pages	Sequential scan of S	K
W	10 page I/Os	100 records on 10 pages	Sequential scan of W	K
RS	...	10K records on 2K pages	...	
SR	...		...	
...				

Space for your answer:

Let's finish the table first:

Subquery	Cost	Size of output	P/K
R	100 page I/Os	1 K records on 100 pages	K
S	1 k page I/Os	10 k records on 1 K pages	K
W	10 page I/Os	100 records on 10 pages	K
RS	100×1000+100 = 100100	10 K records on 2k Pages	K
SR	1000×100+1000 = 101000	10k records on 2k Pages	P
RW	100×10+100 = 1100	100 records on 20 pages	P
WR	10×100+10 = 1010	100 records on 20 pages	K
SW	1000×10+1000 = 11000	1 K records on 200 pages	P
WS	10×1000+10 = 10010	1 K records on 200 pages	K
(RS) W	2k×10+100100 = 120100	1 K records on 300 pages	P
(WR) S	20×1K+1010 = 21010	1 K records on 300 pages	k
(WS) R	200×100+10010 = 30010	1 K records on 300 pages	P

For the output size: RS & SR: $1\text{K} \cdot 10\text{K} \cdot 0.1\% = 10\text{K}$

Since tuple size is doubled, we need 2k pages for this

RW & WR: $1\text{K} \cdot 100 \cdot 0.1\% = 100$

Since tuple size is doubled, we need 20 pages for this

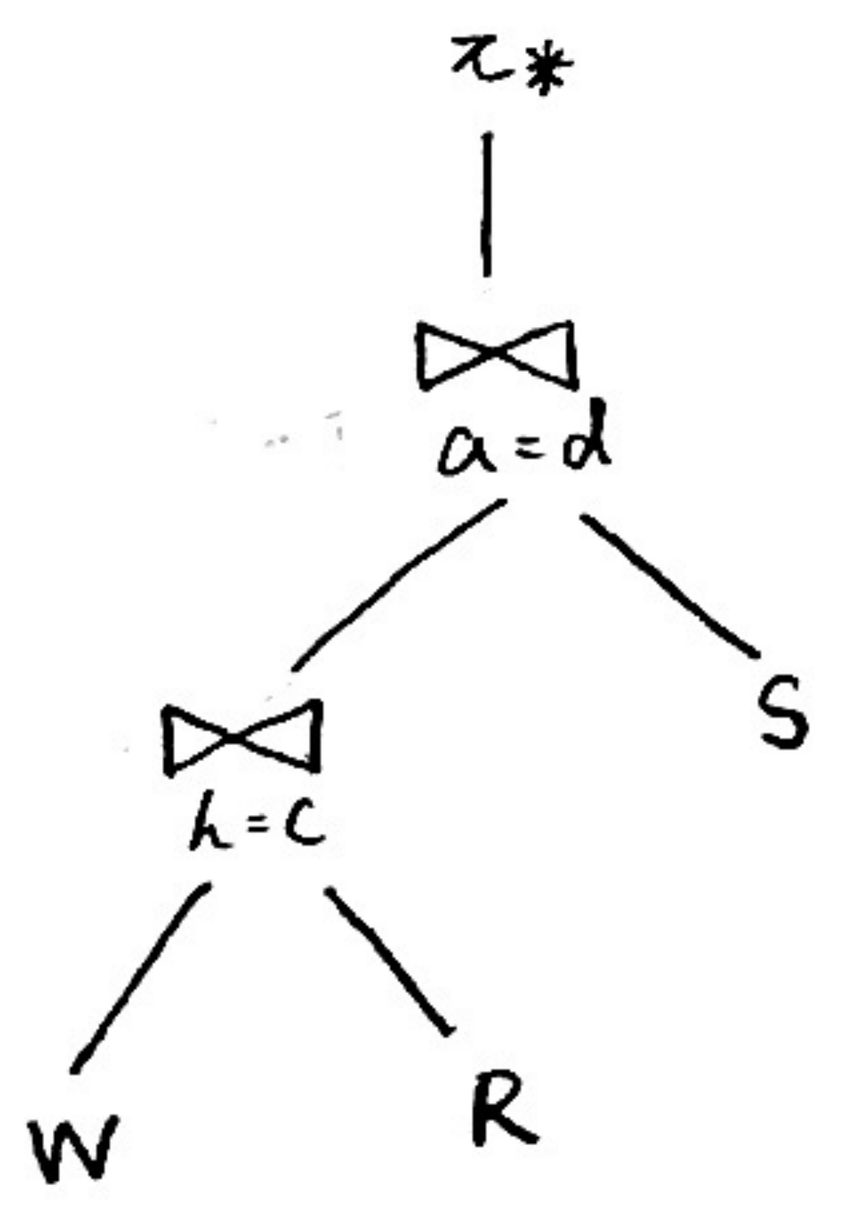
SW & WS: $10\text{K} \cdot 100 \cdot 0.1\% = 1000$

Since tuple size is doubled, we need 200 pages for this

(RS) W & (WR) S & (WS) R : 1 K

and we need 300 pages for the records

Therefore,



(b) (1 point) 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
```

After we added an unclustered B+ tree index on S.d, when we are joining R and S, we could use nested-loop joins instead of sequential scans, which will reduce the cost of reading and writing records. After we added an clustered B+ tree index on R.b, the search speed for R.b > 100 will increase, which will optimize the query plan. Also, the joining of R and S and the filtering might also change the table join order, which will potentially reduce the I/O cost. Therefore, the above modification will likely optimize the query plan.

(c) **(1 point) What is an interesting order? Give one or two examples and explain how they can be useful in getting a better physical query plan.**

one interesting order could be join tables with smaller sizes first. In this case, the intermediate relations will have lower cardinalities. Assuming the same selectivity, the cardinality of joining 100 and 100 records is way smaller than joining 200 and 200. The lower cardinality will then help us reduce the cost for the join.

Another interesting order could be sorting records. In this case, searching up and mapping different records will be faster. Just like R.b we had earlier, the sorted records can speed up the condition of R.b > 100.

Therefore, these two orders could help to optimize the query plan.