Submitted By:
Aman Pandita

DBMS Final Exam

panditaa@oregonstate.edu

Question 1:

1. Block Nested Loop: the cost is:
   (40,000 + (40,000 / (405 - 1)) * 400
   Cost = 39603 I/Os.

2. Sort-Merge: For sort-merge join, the cost:
   Cost = 5B(R) + 5B(S) = 5(40000) + 5(400)
   Cost = 2,02,000 I/Os.

3. Page oriented nested join:
   Cost = B(R) + B(R)*B(S) = 40000 + 40000*400
   Cost = 1,60,40,000 I/Os

Therefore, The Block nested loop will be the one with the least number of I/Os compared to others as it loads blocks of data instead of individual pages, hence reducing the number of I/Os Operations.

Question 2:

When compared to block nested loop or sort-merge join methods, the hash join technique frequently takes less average I/O requests. It may also be adjusted to compute a left anti-join. This benefit is mostly attributable to the efficient utilization of memory by a hash join in reducing disk I/O operations.
To understand this, let's break down the steps in a hash join algorithm and see how it can be modified for a left anti-join:

1. **Hashing Phase:** In this stage, the join attribute (in this example, attribute B) is used to hash the tuples of the smaller relation (S) that have been read into memory. A hash table is used to store the outcomes. This procedure often includes a single, efficient I/O operation—a sequential read—of the smaller relation.

2. **Probe Phase**: The hash join would typically traverse the tuples of the bigger relation (R) at this stage, hash the join attribute, and search for matches in the hash table. Finding tuples in R without a match in S is the goal of a left anti-join, though. As a result, we change the probe phase to only output a tuple from R if the hash table does not contain a matching tuple. Once again, the bigger relation is read sequentially during this operation.

By retaining the smaller relation in memory and sequentially scanning the bigger relation just once, this modified hash join technique often performs left anti-join operations quicker than block nested loop or sort-merge join algorithms.

Example:

Relation R (A, B):

1, 10
1, 20
2, 30
2, 40

Relation S (B, C):

10, 75
10, 85
30, 95

Here, relation S is smaller than R.

Step 1: Hash relation S

Creating a hash table based on the attribute B from relation S:
10 → (10, 75), (10, 85)

30 → (30, 95)

Step 2: Scan through relation R

Now, for each tuple in relation R and check for a matching tuple in the hash table

we find a match in the hash table for tuple (1, 10)
no match is found in the hash table for tuple (1, 20), (We output this tuple)
we find a match in the hash table for tuple (2, 30).
no matches are found in the hash table for (2, 40). (We output these tuple.)

Due to the fact that these tuples from relation R did not have a corresponding tuple in relation S, the result of the left anti-join operation will be (1, 20), (2, 40). Because seeking up items in a hash table is an O(1) operation, it may be completed in a constant average time.

Question 3:
a.

When it comes to creating result tuples, the Block Nested Loop join and Sort-Merge join algorithms offer varying degrees of efficiency. The Block Nested Loop join may begin producing results as soon as it begins working with the two input relations, in contrast to the Sort-Merge join, which demands that both relations be fully sorted before the join operation is carried out.

Block Nested Loop join might swiftly produce the first 10 tuples if the data distribution is balanced and the least desirable condition is avoided (where the necessary tuples are at the very tail end of the relation). The method can begin streaming S and producing output tuples as soon as one block of R has been loaded into memory.

On the other hand, Sort-Merge joins demand that both R and S be sorted beforehand. The amount of time and I/O operations required might vary, mostly based on the particular sorting algorithm employed and the distribution of the data. No output tuples can be created until this sorting procedure has been finished.

In essence, the Block Nested Loop join is expected to need less I/O operations on average than a Sort-Merge join for creating a small set of results, say 10 tuples.

b.

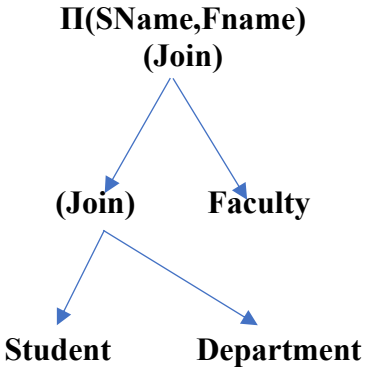
By combining the merge step of sorting with the merge stage of the join operation, the efficiency of the Sort-Merge join algorithm might be improved in this situation. As a result, the join operation no longer requires a separate merging stage, which lowers the I/O cost related to reading and writing the sorted relations.

This change enables the creation of output tuples when matched tuples are found, right during the merging procedure. As a consequence, we can start producing results even before both relations R and S have been fully sorted. Compared to the traditional Sort-Merge join, which requires complete sorting of both relations before creating any output tuples, this improved Sort-Merge join may provide the requisite 10 tuples more quickly.

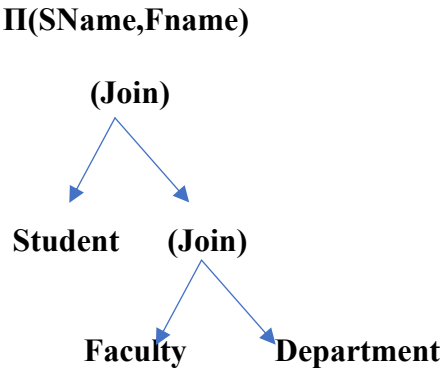
The exact data distribution will determine whether this improved strategy outperforms the Block Nested Loop join. The Block Nested Loop join may still be superior if the required tuples are found early in the process since it may start producing results right away, whereas even the optimized Sort-Merge join requires some sorting.

Question 4:

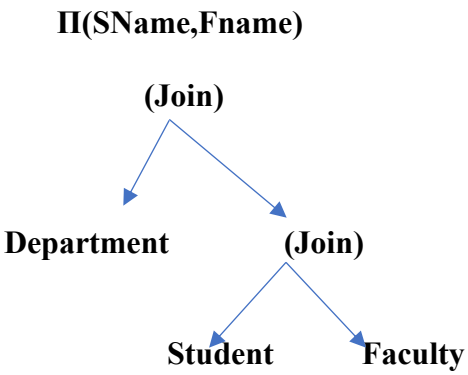
- **(Student ⋈ Department) ⋈ Faculty**



- **(Department ⋈ Faculty) ⋈ Student**



- **(Student ⋈ Faculty) ⋈ Department**



Question 5:

Tuples in Relation R contains 20,000 tuples

number of distinct values in attribute A, V(R,A) = 500

two conditions required:

A=20 and (1 < B < 10).

Calculating for A=20: selectivity(A=20) = 1 / V(R, A) = **1 / 500**.

Calculating for **(1 < B < 10):**

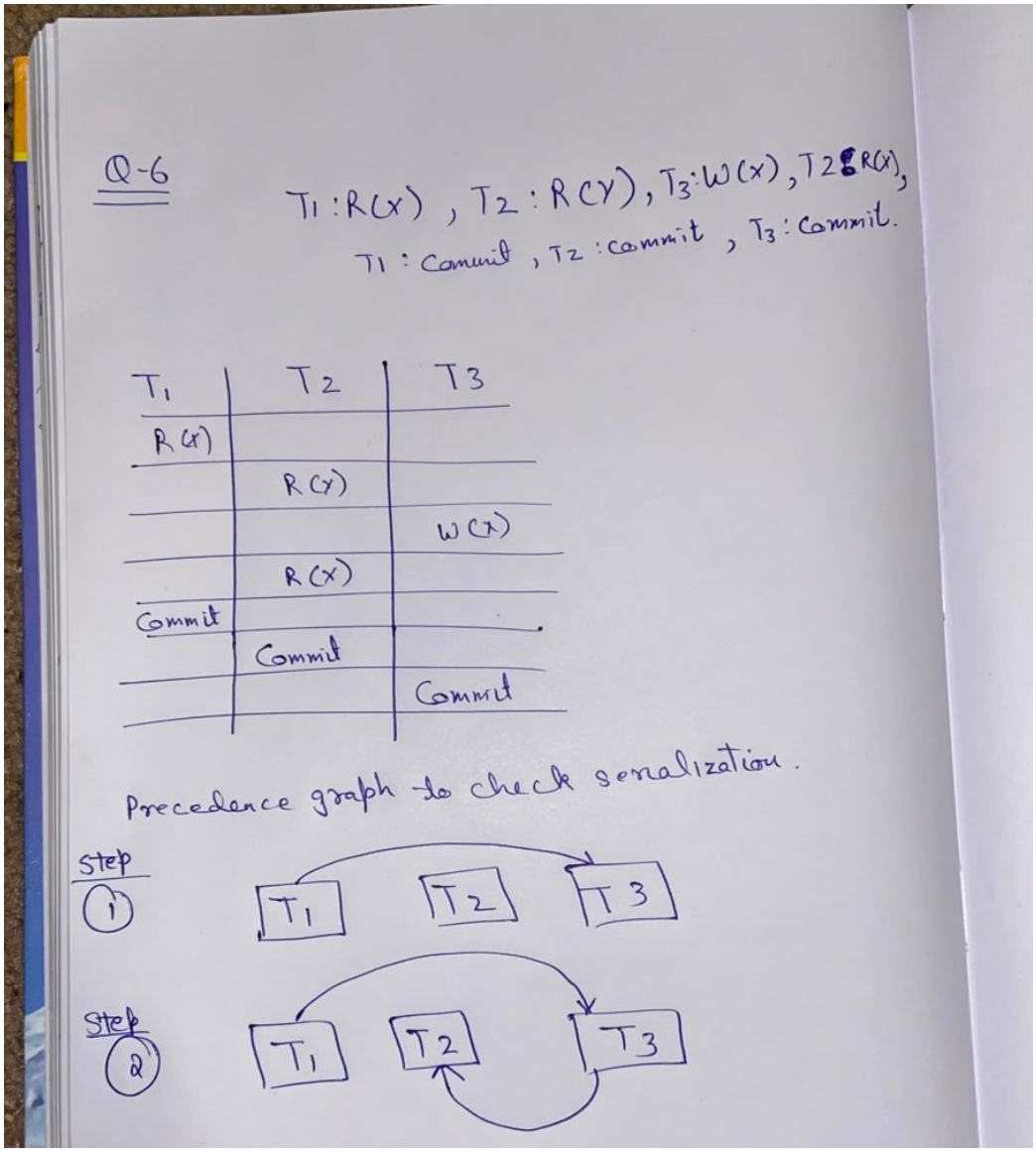
We will use a predefined magic number of 1/4 to estimate the selectivity.

THEREFORE, Estimated size of U = selectivity(A=20) * selectivity(1 < B < 10) * |R|

= 1 / 500 * 1/4 * 20,000 = 10.

Therefore, the estimated size of the result U is **10 tuples.**

Question 6:



Resultant Serialization: **T1 → T3 → T2**

We can see that there is no cycle or loop in it hence it is **serializable**.

Question 7:

T1	T2
W(x)	
	W(X)
Commit	
	R(X)
	Commit

Serialization Graph

The maximum degree of consistency for T2 in the above schedule can be 1 because it requires the value written by T1 and also does not have any committed changes.

Hence due to the requirement of the value from T1's write operation without any committed changes, we can safely say that this proves the consistency to be 1.