Indian Institute of Information Technology Allahabad
End-Sem Solution Paper

Course Name: ***Database Management Systems*** **Course Instructor/ Coordinator:** **Dr. S. MAITY**

Course Code: ***PC-IT-DBM404*** **Program Name(s):** ***BTech(IT) & BTech(IT-Bin)***

Exam Date: ***07-May-2024*** **MM:** ***40***

Q1. Consider the following schedule for transactions T1, T2 and T3: **2 Marks**

T1	T2	T3
Read (X)		
	Read (Y)	
		Read (Y)
	Write (Y)	
Write (X)		
		Write (X)
	Read (X)	
	Write (X)	

Find out the correct serialization order of the transactions in the above schedule. Justify your answer, why so.
Solution: T1->>T3->>T2,
T1 can complete before T2 and T3 as there is no conflict between Write(X) of T1 and the operations in T2 and T3 which occur before Write(X) of T1 in the above diagram.
T3 should can complete before T2 as the Read(Y) of T3 doesn't conflict with Read(Y) of T2. Similarly, Write(X) of T3 doesn't conflict with Read(Y) and Write(Y) operations of T2.
Another way to solve this question is to create a dependency graph and topologically sort the dependency graph. After topologically sorting, we can see the sequence T1, T3, T2.

Q2. Consider the transactions T1, T2, and T3 and the schedules S1 and S2 given below.

T1: r1(X); r1(Z); w1(X); w1(Z)
T2: r2(Y); r2(Z); w2(Z)
T3: r3(Y); r3(X); w3(Y)
S1: r1(X); r3(Y); r3(X); r2(Y); r2(Z);
w3(Y); w2(Z); r1(Z); w1(X); w1(Z)
S2: r1(X); r3(Y); r2(Y); r3(X); r1(Z);
r2(Z); w3(Y); w1(X); w2(Z); w1(Z)

1) Test the conflict serializability of the above schedules. **2 Marks**
Solution: Only ~~S1 is conflict-serializable but~~ not S2.
2) Draw the precedence graph for both schedules. **1 Marks**
Solution:

Schedule S1:

T1	T2	T3
r(X)		
		r(Y)
		r(X)
	r(Y)	
	r(Z)	
		w(Y)
	w(Z)	
r(Z)		
w(X)		
w(Z)		

$$\begin{tikzcd} & T_1 & \\ T_3 \arrow[ur] & & T_2 \arrow[ul] \arrow[ll] \end{tikzcd}$$

No cycle here, therefore conflict serializable.

S2:

T1	T2	T3
r(X)		
		r(Y)
	r(Y)	
		r(X)
r(Z)		
	r(Z)	
		w(Y)
w(X)		
	w(Z)	
w(Z)		

Precedence graph of S2:

$$\begin{tikzcd} & T_1 \arrow[dr, bend left] & \\ T_3 \arrow[ur] & & T_2 \arrow[ul] \arrow[ll] \end{tikzcd}$$

- So there is loop in S2 so S2 is not conflict serializable.

3) The schedule which is conflict serializable, write the serializability order of the transactions for that schedule. **2 Marks**

Solution: The serial order for S1 is T2 -> T3 -> T1.

Q3. (a) Explain about two phase locking (2PL) protocol. **1 Marks**

Solution: A transaction is said to follow the Two-Phase Locking protocol if Locking and Unlocking can be done in two phases.

Growing Phase: New locks on data items may be acquired but none can be released.

Shrinking Phase: Existing locks may be released but no new locks can be acquired.

~~Initially, a~~ transaction is in the growing phase. The transaction acquires locks as needed. Once the transaction releases a lock, it enters the shrinking phase, and it can issue no more lock requests. The two-phase locking protocol ensures conflict serializability.

(b) For the schedule given below

T1	**T2**
Read A	
	Read B
Write A	
	Read A
	Write A
	Write B
Read B	
Write B	

1. Find whether the above schedule is serializable or not? Justify your answer. **1 Marks**
   Solution: This schedule is not serialisable. There are conflicting instruction in T1 and T2, Initial read of data item A is done by T1 whereas final write of data item B is done by T1. therefore W1(A) , R2(A) and W2(B) , R1(B) are conflicting pairs. The schedule is neither T1 –> T2 nor T2 –> T1 serializable.
2. Also find whether it can occur in the 2PL locking protocol? Justify your answer. **1 Marks**
   Solution: Since schedule is not serializable , it can not occur in ~~scheme using~~ 2PL protocol .

Q4. Explain view serializability. In the given schedule, check whether it is view serializable or not. Also check whether the schedule is conflict equivalent to <T1, T5>. In both cases, justify your answer. **2 Marks**

T_1	T_5
read(A)	
$A := A - 50$	
write(A)	
	read(B)
	$B := B - 10$
	write(B)
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$A := A + 10$
	write(A)

Solution:
Consider two schedules S and S′, where the same set of transactions participates in both schedules. The schedules S and S′ are said to be view equivalent if three conditions are met:

1. For each data item Q, if transaction T_i reads the initial value of Q in schedule S, then transaction T_i must, in schedule S′, also read the initial value of Q.
2. For each data item Q, if transaction T_i executes read(Q) in schedule S, and if that value was produced by a write(Q) operation executed by transaction T_j, then the read(Q) operation of transaction T_i must, in schedule S′, also read the value of Q that was produced by the same write(Q) operation of transaction T_j.
3. For each data item Q, the transaction (if any) that performs the final write(Q) operation in schedule S must perform the final write(Q) in schedule S′.

The concept of view equivalence leads to the concept of view serializability. We say that a schedule S is view serializable if it is view equivalent to a serial schedule.

Schedule given in question is not view serializable. Explanation is based on the above three conditions.

Schedule given in question is not conflict equivalent to the serial schedule <T1,T5>, since, in schedule, the write(B) instruction of T5 conflicts with the read(B) instruction of T1. This creates an edge T5 → T1 in the precedence graph. Similarly, we see that the write(A) instruction of T1 conflicts with the read instruction of T5, creating an edge T1 → T5. This shows that the precedence graph has a cycle and that schedule is not serializable.

Q5. During its execution, a transaction passes through several states, until it finally commits or aborts. List all possible sequences of states through which a transaction may pass. Explain why each state transition may occur.

2 Marks

Solution:

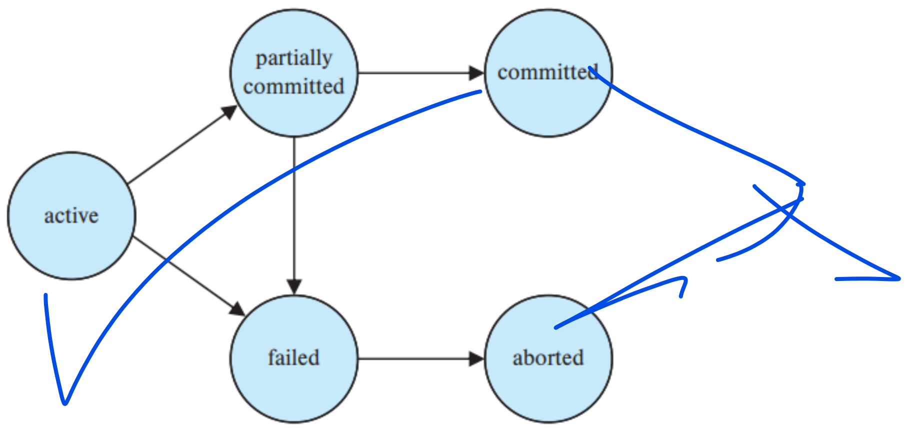


Figure 17.1 State diagram of a transaction.

Possible sequences are:

1. active → partially committed → committed. This is the normal sequence a successful transaction will follow. After executing all its statements it enters the partially committed state. After enough recovery information has been written to disk, the transaction finally enters the committed state.
2. active → partially committed → failed → aborted. After executing the last statement of the transaction, it enters the partially committed state. But before enough recovery information is written to disk, a hardware failure may occur destroying the memory contents. In this case the changes which it made to the database are undone, and the transaction enters the aborted state.
3. active → failed → aborted. After the transaction starts, if it is discovered at some point that normal execution cannot continue (either due to internal program errors or external errors), it enters the failed state. It is then rolled back, after which it enters the aborted state.

Q6. Explain the distinction between the terms serial schedule and serializable schedule.

1 Marks

Solutions: A schedule in which all the instructions belonging to one single transaction appear together is called a serial schedule.

A serializable schedule has a weaker restriction that it should be equivalent to some serial schedule.

2 Marks

——---

1½

Q.7 Most implementations of database systems use strict two-phase locking. Suggest three reasons for the popularity of this protocol.

- Low Rollback Overhead: Two-phase locking typically results in cascadeless schedules, meaning that transactions can be rolled back without causing cascading rollbacks in other transactions. This reduces the overhead associated with transaction rollbacks, as only the affected transaction needs to be rolled back without affecting other transactions, thereby improving system efficiency and reliability.
- Acceptable Concurrency Levels: Despite being strict in its locking protocol, two-phase locking usually allows for an acceptable level of concurrency. Transactions can acquire multiple locks in the growing phase, enabling concurrent execution of transactions until they reach their lock point. This balance between strictness and concurrency enables efficient utilization of system resources while maintaining data consistency and integrity.
- Deadlock Prevention: Two-phase locking helps prevent deadlock situations where transactions are mutually blocking each other, leading to a halt in progress. By adhering to a strict protocol of acquiring and releasing locks in two phases, the likelihood of deadlocks is

2

✓ [illegible], [illegible]

1n

minimized.

3 Marks

Q.8 Underwhat conditions is it less expensive to avoid deadlock than to allow deadlocks to occur and then to detect them?

1. Frequent Deadlocks: When deadlocks occur frequently, the overhead of constantly detecting and resolving them becomes significant.
2. Complex Detection and Resolution: If the detection and resolution of deadlocks are complex and resource-intensive, it's more cost-effective to prevent them upfront.
3. High Impact on System Performance: When deadlocks significantly impact system performance, such as resource wastage and reduced throughput, it's preferable to avoid them to maintain system efficiency.

3 Marks

Q.9. If deadlock is avoided by deadlock-avoidance schemes, is starvation still possible? Explain your answer. 1.5

In deadlock-avoidance schemes, starvation is possible due to prioritizing certain processes over others to prevent deadlocks. Resource allocation decisions favor higher-priority processes, potentially leaving lower-priority processes waiting indefinitely. While deadlock avoidance reduces deadlock risk, it doesn't guarantee fair resource allocation or eliminate starvation entirely. Balancing deadlock prevention with fair access is crucial in designing effective avoidance schemes.

1.7

2 Marks

Q.10 In multiple-granularity locking, what is the difference between implicit and explicit locking

3 Marks

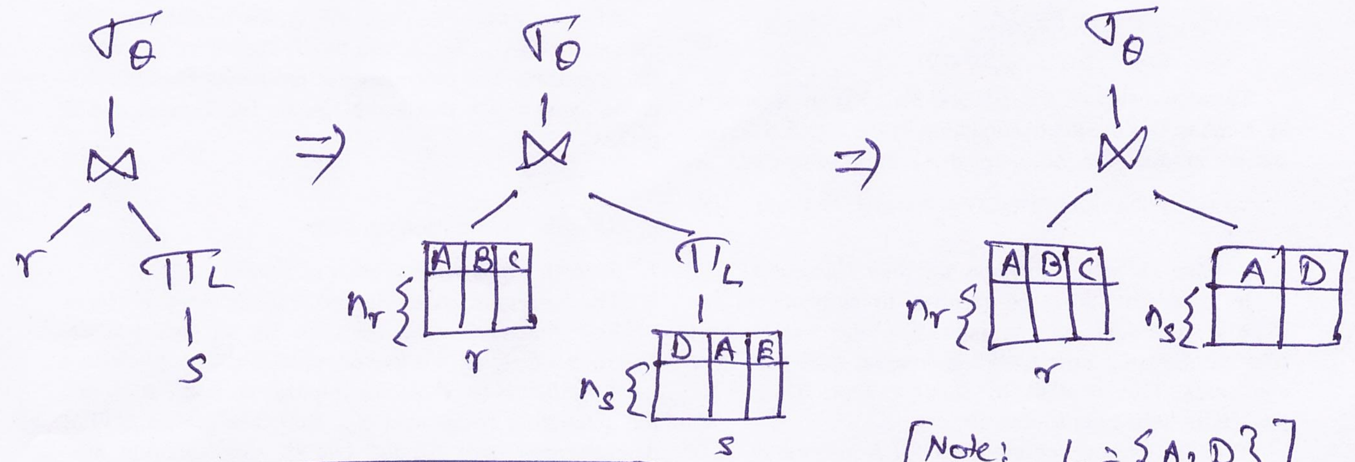
1

Implicit Locking	Explicit Locking
Locks acquired and released automatically by the system based on predefined rules or algorithms.	Users explicitly request locks on specific resources or objects within the database.
Offers less flexibility as the system determines locking granularity.	Provides greater flexibility and control over locking operations for users.
Generally easier to use with minimal user intervention.	Requires users to have a deeper understanding of the database schema and concurrency control mechanisms.

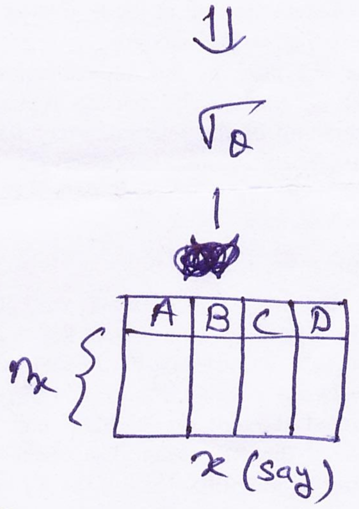
18

Q. 11(A) & 11(C)

We have to take a bottom-up approach:-



[Note: $L = \{A, D\}$]



[Note: $n_x = n_s$ as, the join attr. ('A') is a foreign key in 'S' referencing to 'R'.]

Now, Let,

$$size(\sigma_{\theta_1}(x)) = l_1$$

and $size(\sigma_{\theta_2}(x)) = l_2$

So, we know that:-

$$size(\sigma_{\theta_1 \vee \theta_2}(x))$$

$$= n_x * \left[1 - \left(1 - \frac{l_1}{n_x}\right) * \left(1 - \frac{l_2}{n_x}\right)\right]$$

[see book for more details]

$$= n_s * \left[1 - \left(1 - \frac{l_1}{n_s}\right) \cdot \left(1 - \frac{l_2}{n_s}\right)\right] \quad \text{.... eqn. ①}$$

$l_1 = size(\sigma_{\theta_1}(x)) = size(\sigma_{D > 70}(x))$

$$= n_x * \frac{Max(D, s) - 70}{Max(D, s) - Min(D, s)}$$

$$= n_s * \frac{(100 - 70)}{(100 - 10)}$$

$$= n_s * \frac{30}{90} = \boxed{\frac{n_s}{3}}$$

$$l_2 = \text{size}(\sigma_{\theta_2}(x)) = \text{size}(\sigma_{A=a_i}(x))$$

$$= \frac{n_x}{V(A,x)} = \frac{n_s}{V(A,x)} = \frac{n_s}{V(A,r)}$$

[Note, we are assuming $V(A,x) = V(A,r)$ as we cannot have a better estimate]

$$\boxed{= \frac{n_s}{4}}$$

Replacing the values of l_1 & l_2 in eqn. ① we get:-

$$\text{size}(E) = n_s * \left[1 - \left(1 - \frac{1}{3}\right)\cdot\left(1 - \frac{1}{4}\right)\right]$$

$$\boxed{= \frac{n_s}{2}}$$

Q. 11B

Size of one tuple of x = size(A) + size(B) + size(C) + size(D)

$= 1+2+2+2 = 7$ bytes.

Hence, $\text{size}(E) = 7 * \frac{n_s}{2} = 7 * \frac{80}{\cancel{2}} = 280$ bytes

Q. 11 D The original expression-tree:-

$$\begin{tikzcd}
& \sigma_{\theta_1 \vee \theta_2} \arrow[d, dash] & \\
& \bowtie \arrow[dl, dash] \arrow[dr, dash] & \\
r & & \Pi_{A,D} \arrow[d, dash] \\
& & S
\end{tikzcd}$$

$\sigma_{\theta}(r \bowtie \Pi_{A,D}(s))$

$\equiv \sigma_{A='a' \vee D>70}(r \bowtie \Pi_{A,D}(s))$

The "Heuristic Rules" that we'ld apply:-

① "Perform Selection as early as possible"
② "Perform Projection as early as possible"

} see book for more details

So, the following two (both) are Correct Answers:-

✓
$$\begin{tikzcd}
& \bowtie \arrow[dl, dash] \arrow[dr, dash] & \\
r & & \Pi_{A,D} \arrow[d, dash] \\
& & \sigma_{\theta_1 \vee \theta_2} \arrow[d, dash] \\
& & S
\end{tikzcd}$$

$\equiv$

✓
$$\begin{tikzcd}
& \bowtie \arrow[dl, dash] \arrow[dr, dash] & \\
r & & \sigma_{\theta_1 \vee \theta_2} \arrow[d, dash] \\
& & \Pi_{A,D} \arrow[d, dash] \\
& & S
\end{tikzcd}$$

BUT, THE FOLLOWING ANSWERS WOULD BE INCORRECT!-

X
$$\begin{tikzcd}
& \sigma_{\theta_1} \arrow[d, dash] & \\
& \sigma_{\theta_2} \arrow[d, dash] & \\
& \bowtie \arrow[dl, dash] \arrow[dr, dash] & \\
r & & \Pi_{A,D} \arrow[d, dash] \\
& & S
\end{tikzcd}$$

or

X
$$\begin{tikzcd}
& \bowtie \arrow[dl, dash] \arrow[dr, dash] & \\
\sigma_{\theta_2} \arrow[d, dash] & & \sigma_{\theta_1} \arrow[d, dash] \\
r & & \Pi_{A,D} \arrow[d, dash] \\
& & S
\end{tikzcd}$$

20

Q. 12 A Counter-example to show that "Perform Projection early" - not always good:-

Let, $r(R)$ and $s(S)$

where $R(A, B, C)$ & $S(A, D, E)$

Assume:- ⊙ r is a very small relation compared to s

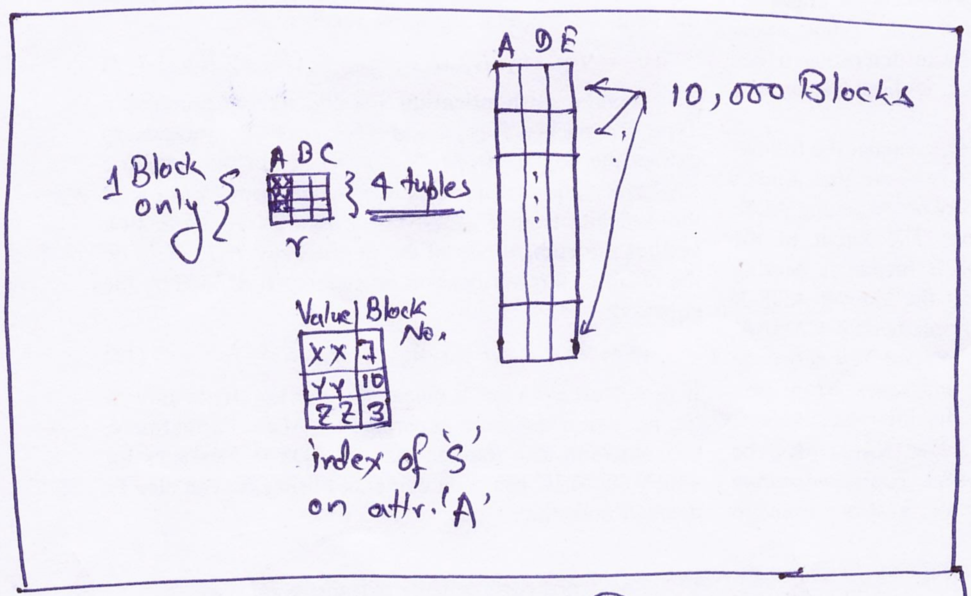
⊙ there is an index on the join attribute 'A' on s

Now, we have to evaluate the following query:-

$$\Pi_{A,D}(r \bowtie s) \quad \ldots \text{①}$$

So, if we apply the heuristic rule the equivalent expression be:-

$$r \bowtie \Pi_{A,D}(s) \quad \ldots \text{②}$$



If we evaluate expression ①

Total no. of block reads required:-

1 — for reading 'r'

1 — for reading the 'index' file

4 - One block of 's' for each tuple of 'r'

Total = 5

⊙ This will be followed by a projection on the result
- requires no extra block access

If we evalue expression ②:-

⊙ We have to access 10,000 blocks from Disk

⊙ this will be followed by the join operation
- which may require some extra block accesses

Question No. 13: Example of use of dynamic programming to reduce the cost of query optimization

For example, suppose we want to find the best join order of the form:

$$(r_1 \bowtie r_2 \bowtie r_3) \bowtie r_4 \bowtie r_5$$

which represents all join orders where r_1, r_2, and r_3 are joined first (in some order), and the result is joined (in some order) with r_4 and r_5. There are 12 different join orders for computing $r_1 \bowtie r_2 \bowtie r_3$, as shown below:-

$r_1 \bowtie (r_2 \bowtie r_3)$	$r_1 \bowtie (r_3 \bowtie r_2)$	$(r_2 \bowtie r_3) \bowtie r_1$	$(r_3 \bowtie r_2) \bowtie r_1$
$r_2 \bowtie (r_1 \bowtie r_3)$	$r_2 \bowtie (r_3 \bowtie r_1)$	$(r_1 \bowtie r_3) \bowtie r_2$	$(r_3 \bowtie r_1) \bowtie r_2$
$r_3 \bowtie (r_1 \bowtie r_2)$	$r_3 \bowtie (r_2 \bowtie r_1)$	$(r_1 \bowtie r_2) \bowtie r_3$	$(r_2 \bowtie r_1) \bowtie r_3$

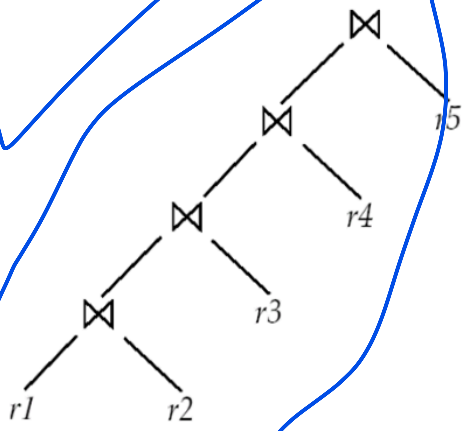
and 12 orders for computing the join of this result with r_4 and r_5. Thus, there appear to be 144 join orders to examine. However, once we have found the best join order for the subset of relations $\{r_1, r_2, r_3\}$, we can use that order for further joins with r_4 and r_5, and can ignore all costlier join orders of $r_1 \bowtie r_2 \bowtie r_3$. Thus, instead of 144 choices to examine, we need to examine only 12 + 12 choices.

২

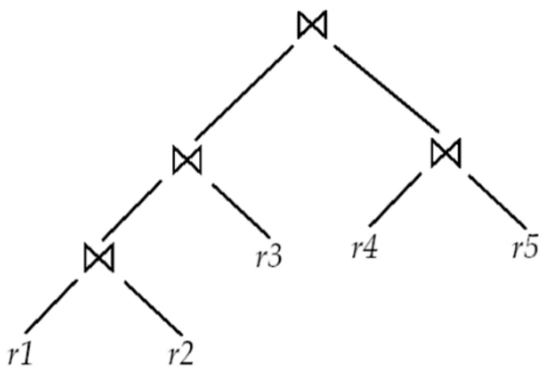
[illegible]

Solution to Question No. 14

- In **left-deep join trees,** the right-hand-side input for each join is a relation, not the result of an intermediate join.



(a) Left-deep join tree



(b) Non-left-deep join tree

In which scenario a query optimizer would prefer to choose left-deep or right-deep trees:

In architectures where parallel processing by multiple processors are allowed / pipelining in the h/w architecture is allowed

advantage of using left-deep or right-deep trees:-

1) Speed-up due to parallel processing (during query execution)
2) Reduces cost of query optimizer
    a) Needs to evaluate only (n)! Join-orders (in left / right deep join trees)
    b) Instead of, (2(n-1))! / (n-1)! Join-orders (without eft / right deep join trees)

৭৩

[illegible]

Q.No.15

$$\Pi_L(E_1 - E_2) = \Pi_L(E_1) - \Pi_L(E_2)$$

Incorrect

All others are correct

Example

A	B
a_1	b_1
a_1	b_2
a_2	b_1

R

A	B
a_1	b_1
a_1	b_3

S

$$(R - S) = \{(a_1, b_2), (a_2, b_1)\}$$

A	B
a_1	b_2
a_2	b_1

R−S

$$\Pi_A(R - S) = \{a_1, a_2\}$$

But $\Pi_A(R) = \{a_1, a_2\}$

$$\Pi_A(S) = \{a_1\}$$

So, $\Pi_A(R) - \Pi_A(S) = a_2$

Hence. $\Pi_A(R - S) \neq \Pi_A(R) - \Pi_A(S)$