**Assignment 2**

**Question1:**

**1):**

NO, .

, therefore .

**2):**

Step1:

Step2:

; thus and are not inferred by . Hence, could not be replaced by or .

and ; thus and are not inferred by .

Hence, could not be replaced by or .

and ; thus is not inferred by but is inferred by . Hence, could be replaced by .

and ; thus is not inferred by but is inferred by . Hence, could be replaced by .

Step3:

; thus is not inferred by . That is is not redundant.

; thus is not inferred by . That is is not redundant.

; thus is not inferred by . That is is not redundant.

; thus is not inferred by . That is is not redundant.

; thus is not inferred by . That is is not redundant.

; thus H is not inferred by . That is is not redundant.

; thus is not inferred by . That is is not redundant.

; thus is not inferred by . That is is not redundant.

Iteratively, we could find that does not change.

Thus,

.

**3):**

Note: row1 and row2 and row2 and row3 of S agree on , which is the left hand side of . Therefore, change the G value on row1 and row3 to a. After that, row1 and row2 and row2 and row3 of S agree on which is the left hand side of , Therefore, change the H value on row1 and row3 to a.

Note: row1 and row3 and row2 and row3 of S agree on , which is the left hand side of . Therefore, change the I value on row1 and row2 to a. After that, row1 and row2 and row2 and row3 of S agree on which is the left hand side of , Therefore, change the J value on row1 and row2 to a. Similarly, we should change the K value on row1 and row2 to a.

Now, row 1 is entirely a’s, so the decomposition is lossless.

4): NO, the reason is below.

is super-key

is super-key, cause

is super-key, cause

is super-key, cause and

is super-key, cause , and

is super-key, cause , , and

5):

Candidate key is {E,I,K}, but BD is not super-key so

For , because and , therefore, BD and E could be super-key. So, is in BCNF

For , Candidate key is {A,D,H,I,J,K}, but H is not super-key so

For , H is the super-key therefore, is in BCNF

For , Candidate key is {A,D,H,I,J,K}, but A is not super-key so

For , A is the super-key therefore, is in BCNF

For , Candidate key is {A,D,H,I,J,K}, which is derived from H but A is not super-key so

For , A is the super-key therefore, is in BCNF

For , is super-key of thus, is in BCNF

The decomposition of R is ,,,, and . In this process, it is lossless-join, however, has been lost so it is not dependency-preserving.

Therefore, it is not possible to decompose 𝑅 into a collection of BCNF relations and ensure the decomposition is dependency-preserving and lossless-join.

**Question2:**



***Figure 1***

a):

As can be shown from figure1, T1 and T3 have not been completed before time 8, therefore, T1 and T3 are undo. T2 has been committed before time so T2 is redo.

b):

T1 and T3 should be undo since the times they consumed exceed time8.

We need not redo T2 as the database updates including T2 in buffers are force-written before Time7.

**Question3:**

**1):**

Date page: p1, p2 and p3

Queries:

Buffer:

|  |  |
| --- | --- |
| p1 | p2 |

**For First in First Out (FIFO) buffer replacement policy:**

In Q3, p3 would be introduced and p1 would be replaced as p1 is first in.

|  |  |
| --- | --- |
| p3 | p2 |

Buffer would become

Then in Q4 and Q5, system would directly read p2 and p3. In each loop, system would also directly read p3 and p2.

**For Most Recently Used (MRU) buffer replacement policy:**

In Q3, p3 would be introduced and p2 would be replaced since it is mostly recently used.

Buffer:

|  |  |
| --- | --- |
| p1 | p3 |

Then in Q4, p2 would be introduced and p3 would be replaced since p3 become the new one which is mostly recently used.

|  |  |
| --- | --- |
| p1 | p2 |

Buffer would become

In Q5, p3 would be introduced and p2 would be replaced since p2 become the new one which is mostly recently used.

|  |  |
| --- | --- |
| p1 | p3 |

Buffer would become

In this loop, for Most Recently Used (MRU) buffer replacement policy, the cache miss would generate and the system would need to repeatedly read p2 and p3 in hard disk, thus, the buffer would introduce p2 and p3 repeatedly. This would take much time.

Therefore, First in First Out (FIFO) buffer replacement policy is better than Most Recently Used (MRU) buffer replacement policy in this scenario.

**2):**

Date page: p1, p2 and p3

Queries:

Buffer:

|  |  |
| --- | --- |
| p1 | p2 |

**For First in First Out (FIFO) buffer replacement policy:**

In Q3, p1 would be directly read and the buffer is consistent. In Q4, p3 would be introduced and the p1 would be replaced as it is first in.

|  |  |
| --- | --- |
| p3 | p2 |

Buffer would become

Then in Q5, p2 would be directly read and in Q6, p1 would be introduced and p2 would be replaced. The buffer would become

|  |  |
| --- | --- |
| p3 | p1 |

In Q7, p1 would be directly read.

Therefore, in each loop, the hard disk is only accessed once.

**Least Recently Used (LRU) buffer replacement policy:**

In Q3, p1 would be directly read and the buffer is consistent but p2 would become the one which is least recently used. In Q4, p3 would be introduced and the p2 would be replaced as it is least recently used.

|  |  |
| --- | --- |
| p1 | p3 |

Buffer would become

|  |  |
| --- | --- |
| p2 | p3 |

Then in Q5, p2 would be introduced and p1 would be replaced, the buffer would become

and in Q6, p1 would be introduced and p3 would be replaced. The buffer would become

|  |  |
| --- | --- |
| p2 | p1 |

In Q7, p3 would be introduced and p2 would be replaced. The buffer would become

|  |  |
| --- | --- |
| p3 | p1 |

Therefore, in each loop, the hard disk is accessed thrice. This method speeds more much than FIFO in this scenario.

As a result, First in First Out (FIFO) buffer replacement policy is better than Least Recently Used (LRU) buffer replacement policy in this case.