Question 1

1) No. According to the set F = {A→BC, E→AD, BD→E, CE→DH, H→G, EI→J}, the enclosure set of C is C^+ = {C}
Proving that "C→J" does not belong to F^+.

2) R = (A, B, C, D, E, G, H, I, J, K)
Set F = {A→BC, E→AD, BD→E, CE→DH, H→G, EI→J}
Step 1: F' = {A→B, A→C, E→A, E→D, BD→E, CE→D, CE→H, H→G, EI→J}
Step 2:
BD→E
D^+ = {D} Hence, BD→E cannot be replaced by D→E.
B^+ = {B} Hence, BD→E cannot be replaced by B→E.
CE→D
E^+ = {E, A, B, C, D, H, G} Hence, CE→D can be replaced by E→D.
CE→H
E^+ = {E, A, B, C, D, H, G} Hence, CE→H can be replaced by E→H.
EI→J
E^+ = {E, A, B, C, D, H, G} Hence, EI→J cannot be replaced by E→J.
I^+ = {I} Hence, EI→J cannot be replaced by I→J.
F"= {A→B, A→C, E→A, E→D, E→H, BD→E , H→G, EI→J}
Step 3:
$A^+|_{F''-\{A\to B\}}$ = {A, C} Thus, A→B is not inferred by $A^+|_{F''-\{A\to B\}}$. So, A→B is not redundant.
$A^+|_{F''-\{A\to C\}}$ = {A, B} Thus, A→C is not inferred by $A^+|_{F''-\{A\to C\}}$. So, A→C is not redundant.
$E^+|_{F''-\{E\to A\}}$ = {E, D, H, G} Thus, E→A is not inferred by $E^+|_{F''-\{E\to A\}}$. So, E→A is not redundant.
$E^+|_{F''-\{E\to D\}}$ = {E, A, B, C, H, G} Thus, E→D is not inferred by $E^+|_{F''-\{E\to D\}}$. So, E→D is not redundant.
$E^+|_{F''-\{E\to H\}}$ = {E, A, B, C, D} Thus, E→H is not inferred by $E^+|_{F''-\{E\to H\}}$. So, E→H is not redundant.
$BD^+|_{F''-\{BD\to E\}}$ = {B, D} Thus, BD→E is not inferred by $BD^+|_{F''-\{BD\to E\}}$. So, BD→E is not redundant.
$H^+|_{F''-\{H\to G\}}$ = {H} Thus, H→G is not inferred by $H^+|_{F''-\{H\to G\}}$. So, H→G is not redundant.
$EI^+|_{F''-\{EI\to J\}}$ = {E, I, A, B, C, D, H, G} Thus, EI→J is not inferred by $EI^+|_{F''-\{EI\to J\}}$. So, EI→J is not redundant.
∴ F'''= {A→B, A→C, E→A, E→D, E→H, BD→E , H→G, EI→J}

3) R1 = {ABCDE}, R2 = {EGH}, R3 = {EIJK}.
Therefore, we drew this table for each distinguish variable "a" shows in R1, R2 and R3.

	A	B	C	D	E	G	H	I	J	K
R1	a	a	a	a	a					
R2					a	a	a			
R3					a			a	a	a

We can see from the table above, **the column E has all distinguish variables show in R1, R2 and R3.** Therefore, we will seek for the set "F" if it has **E in the left-hand side**.
And the F is {A→BC, E→AD, BD→E, CE→DH, H→G, EI→J}.
So, we have E→AD with E in the left-hand side, then we add up distinguish variables for column A and column D, that is:

	A	B	C	D	E	G	H	I	J	K
R1	a	a	a	a	a					
R2	a			a	a	a	a			
R3	a			a	a			a	a	a

Iteratively, we will also seek for A and D if either of them shows in the left-hand side in F. Then we have A→BC, so we fill up this table.

	A	B	C	D	E	G	H	I	J	K
R1	a	a	a	a	a					
R2	a	a	a	a	a	a	a			
R3	a	a	a	a	a			a	a	a

Iteratively, we will seek for other available distinguish variables. Consider CE→DH, H→G, then we will have:

	A	B	C	D	E	G	H	I	J	K
R1	a	a	a	a	a	a	a			
R2	a	a	a	a	a	a	a			
R3	a	a	a	a	a	a	a	a	a	a

It is obvious that in the **row R3**, this row is filled up with all "a" (distinguish variables) in its all columns, which proves that the decomposition of R1 = {ABCDE}, R2 = {EGH}, R3 = {EIJK} of R **is lossless join**.

4) Super-keys are the identifiers to determine other attributes in this relational database. So, we could have such five super-keys:
(E, I, K)
(E, A, I, K)
(E, A, I, J, K)
(E, D, I, J, K)
(E, H, I, K)

5) To satisfy the 3 conditions: ① BCNF, ② dependency-preserving, ③ lossless-join decomposition, we firstly make lossless-join and dependency-preserving decompositions, which should satisfy 3NF:

From part 2) we have the minimal cover:

F’”= {A→B, A→C, E→A, E→D, E→H, BD→E , H→G, EI→J}
Therefore, we have such decompositions:

Decompositions	Inferred from
R1 = {A, B, C}	A→B, A→C
R2 = {E, A, D, H}	E→A, E→D, E→H
R3 = {B, D, E}	BD→E
R4 = {H, G}	H→G
R5 = {E, I, J}	EI→J
R6 = {E, I, K}	Nothing to infer K, but E, I, K is super key

Obviously, the decompositions are **dependency-preserving** since all of which are inferred from the minimal cover. Lossless join can also be proved by proving R6 has all distinguish variables (because only R6 has K).

From part 3) we proved {EIJK} is lossless join with {ABCDE} and {EGH}. In these decompositions, E can infer any components in R1, R4, and each of R2, R3, R5, R6 has E, so it can be proved that these decompositions are **lossless join**.

~~As dependency-preserving should be followed and R3 = {B, D, E} cannot be further divided, this relation **should always exist** in decompositions. However, consider E→A and A→B, which means that B is transitively dependent on E, violating the criteria of BCNF. As a result, it is **NOT possible** to satisfy the 3 conditions: ①BCNF, ②dependency-preserving, ③ lossless-join at the same time.~~

This decomposition satisfies BCNF. Although R3 = {B, D, E} cannot be further divided for dependency-preserving, the function E→A and A→B do **NOT** contribute to **Transitive Dependency** in R3, because the relation A does not occur in R3, then there is no transitive link between E and B in R3.

Question 2

1) According to the scheduled timetable, the transactions of T1, T2, T3 will have such tasks respectively:

T1	Read at Time 1, Time 4 and Time 10
T2	Read at Time 3 and Time 5
T3	Read at Time 2, Time 6 and Time 9

Therefore, when we assume that the system crashes at the time 8, T2 has done its tasks. Also, as we assume that a transaction will be committed immediately after all the read/write operations of it are done, T2 has been updated in the database, so we should **REDO T2**. As for T1 and T3, they do not complete their tasks as they will read after Time 8, therefore, we should **UNDO T1** and **UNDO T3**. All recovery operations are showed below:

T1	UNDO
T2	REDO
T3	UNDO

2) If a check point added on the Time 7, every transaction has been done **before Time 7** should NOT be recovered (UNDO or REDO). However, any transactions will be done **after Time 7** still need to be recovered.
That means **T2**, which has been done at Time 5, **should not have REDO or UNDO**, however, **T1** and **T3** still need to have **UNDO**. All recovery operations are showed below:

T1	UNDO
T2	No operations needed, T2 has already been reflected into the database.
T3	UNDO

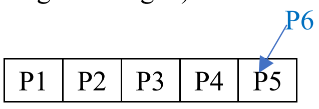
Question 3

1) Here is a scenario that First in First Out (FIFO) buffer replacement policy is better than Most Recently Used (MRU) buffer replacement policy:

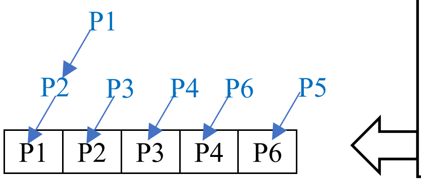
   Suppose we have the data pages: P1, P2, P3, P4, P5, P6
   Suppose the buffer space is: 5
   And suppose we have the two queries:
   Q1: Read P1, P2, P3, P4, P5, P6
   Q2: Read P5, P6, P4, P3, P2, P1 (Reversed sequence of reading, and exchanged the first two)

   If the Most Recently Used (MRU) buffer replacement policy has been used, it will be like this:

   MRU (7 Pages Changed)
   Q1:



   Q2:



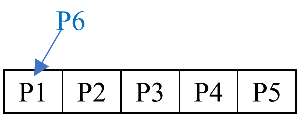
Q2: Read P5, P6, P4, P3, P2, P1
Firstly, P5 does not exist in buffer, change P6 to P5. After that, P6 does not exist in buffer, change P4 to P6… and so on

   According to MRU, in Q1, we release the most recently used frame P5, and replace it with P6. Then in Q2, we firstly check if P5 exists in buffer. As P5 does not exist (recently has been replaced), we release P6 and replace it with P5. After that, we check if P6 exists and so on.

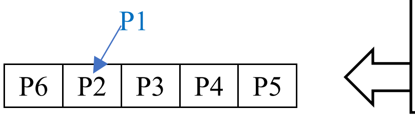
   However, FIFO in this scenario is more efficient as showed below:

   FIFO (2 Pages Changed)

   Q1:



   Q2:



Q2: Read P5, P6, P4, P3, P2, P1
They already exist in buffer **except P1**, therefore we just need to replace P2 with P1 **in the end**.

   According to FIFO, in Q1, we release the first-in frame P1, and replace it with P5. Then in Q2, we firstly check if P5 exists in buffer, and it exists indeed. Iteratively, we check P6, P4, P3, P2 which already exist in buffer. Finally, we just need to replace P2 (the first-in frame) with P1.

   In conclusion, we can see MRU results in 7 pages changing but FIFO needs only 2 pages changing, therefore FIFO is better than MRU buffer replacement policy in this scenario.

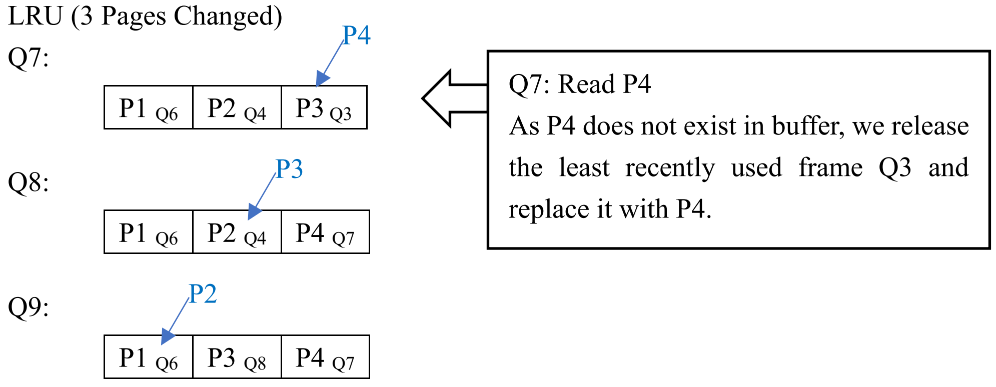
2) Here is a scenario that First in First Out (FIFO) buffer replacement policy is better than Least Recently Used (LRU) buffer replacement policy:

Suppose we have the data pages: P1, P2, P3, P4
Suppose the buffer space is: 3
And suppose we have the such queries:
Q1: Read P1; Q2: Read P2; Q3: Read P3;
Q4: Read P2; Q5: Read P1; Q6: Read P1;
Q7: Read P4; Q8: Read P3; Q9: Read P2

Until Q6, **P1 has been used 3 times, P2 has been used twice, P3 has been used only once.**

If the Least Recently Used (LRU) buffer replacement policy has been used, from Q7 to Q9, it will be like this:

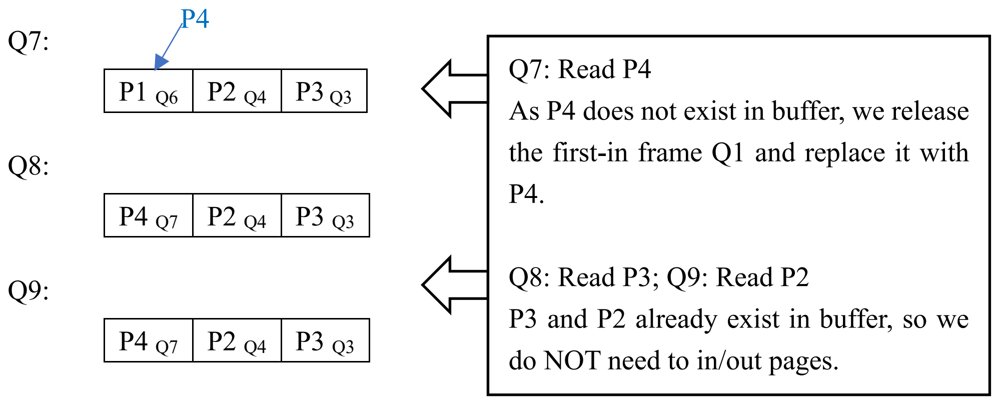
LRU (3 Pages Changed)



Q7: Read P4
As P4 does not exist in buffer, we release the least recently used frame Q3 and replace it with P4.

However, FIFO in this scenario is more efficient as showed below:

FIFO (1 Page Changed)



Q7: Read P4
As P4 does not exist in buffer, we release the first-in frame Q1 and replace it with P4.

Q8: Read P3; Q9: Read P2
P3 and P2 already exist in buffer, so we do NOT need to in/out pages.

In conclusion, we can see LRU results in 3 pages changing, but FIFO needs only 1 page changing, therefore FIFO is better than LRU buffer replacement policy in this scenario.