Coursework 3

Question 1

I will first build the minimum cover for the schema and use the result to derive candidate keys.

Part 2

Firstly, we consider all attributes on the left side of each rule removing one by one and applying $Closure_F(x)$, denoted as $C_F(x)$, to each subset to determine if the attribute is redundant or not. Let F' denote the modified set of functional dependencies which will get altered as we go through the rules. Initially, let F' = F, where F is the set of functional dependencies in question.

Rule	Closure	Action
A B → C F	$C_{F'}(A)$ = A B C D E F G $C_{F'}(B)$ = B	We remove B and get **A → C F**
B G → C	$C_{F'}(B)$ = B $C_{F'}(G)$ = G	We cannot remove either
A E F → C	$C_{F'}(AE)$ = A B C D E F G $C_{F'}(AF)$ = A B C D E F G	We can remove either E or F **A E → C** or **A F → C**
A B G → D E	$C_{F'}(AB)$ = A B C D E F G $C_{F'}(AG)$ = A B C D E F G	We can remove either B or G **A B → D E** or **A G → D E**
C F → A E	$C_{F'}(C)$ = C $C_{F'}(F)$ = F	We cannot remove either
A → C G	$C_{F'}(A)$ = A B C D E F G	Single attribute, cannot be removed at this stage
A D → E F G	$C_{F'}(A)$ = A B C D E F G $C_{F'}(D)$ = D	We can remove D **A → E F G**
A C → B	$C_{F'}(C)$ = C	We can remove C **C → B**

Our modified F' is *A→ CF, BG→ C, AE→ C, AB→ DE, CF→ AE, A→ CG, A→ EFG, A→ B*.
In the second phase we attempt removing attributes from the right hand side and apply closure to the left hand side to determine if the attribute is redundant or not.

Rule from F'	Modified rule	Closure	Action

A → C F	A → F	$C_{F'}$(A) = A B C D E F G	Remove C
A → *F*	A → ∅	$C_{F'}$(A) = A B C D E F G	Remove F
BG → C	BG → ∅	$C_{F'}$(BG) = BG	No action
AE → C	AE → ∅	$C_{F'}$(AE) = A B C D E F G	Remove C
A → CG	A → C	$C_{F'}$(A) = A B C D E F G	Remove C
A → C	A → ∅	$C_{F'}$(A) = A B C D E F G	Remove C
A → EFG	A → FG	$C_{F'}$(A) = A B C D E F G	Remove E

We end up with the following rules which are the minimum cover of the schema:

- **BG → C**
- **AB → DE**
- **CF → AE**
- **A → BFG** (combined from A → FG and A → B)

Part 1

Given the minimum cover from Part 2, we can find the candidate keys but running closure on each of the functional dependencies in the minimum cover.

	FD	**Closure**	**Candidate Key?**
(1)	BG → C	B C G	no
(2)	AB → DE	A B C D E F G	no, (4) is smaller set
(3)	CF → AE	A B C D E F G	yes
(4)	A → BFG	A B C D E F G	yes

Since (2) contains A on its left hand side, we can discard it as a candidate key.
Therefore, we obtain the following candidate keys **{ A, CF, BFG }** where BFG comes from not taking A as a candidate key but taking its right hand side.

Part 3

Given the minimum cover in Part 2, we can find 3NF decomposition. We take X→ Y and find a set Z = X ∪ Y and verify it contains a candidate key from Part 1.

We get:

- AC for A → C,
- ABCE for E → ABC
- CDF for F → CD

Part 4:

Question 2

True.

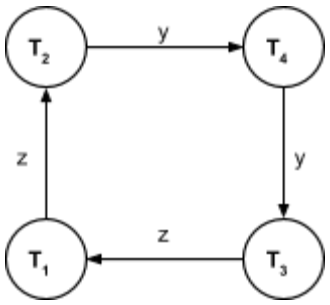
Part → :
Assume there is an attribute P that is prime and belongs to at least one antikey A. Then taking the superset of A will contain a proper superset with A = P. This is a contradiction by the definition of antikey.

Part ←:
Assume the attribute belongs to an antikey and is prime P at the same time. Adding any attribute that is not an element of the antikey will make the new set of attributes a key. However, it will not be a proper superset of the antikey as P would have been in the set originally. This is a contradiction.

Question 3

In order to determine conflict serializability, we can draw a graph with edges from T_x to T_y for operations O_i , O_j where at least one of O_i , O_j is a *write* operation.

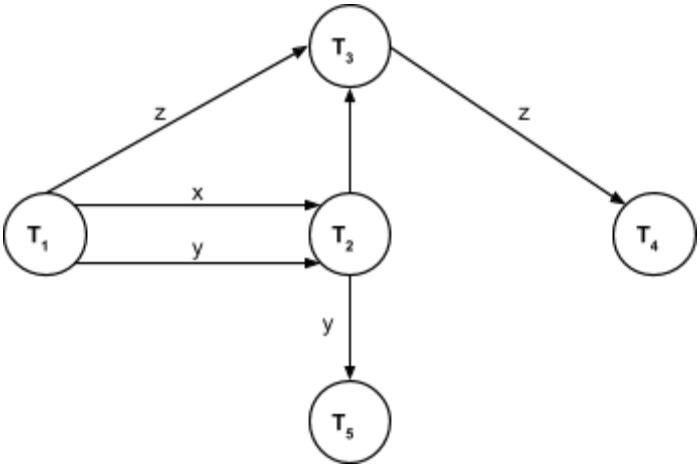
For schedule 1 we obtain the following graph:



We obtain a cycle in graph due to clashes amongst
read(Y) in T_2, *write(Y)* in T_4 followed by *read(Z)* in T_3, *write(Z)* in T_1 and *read/write(Z)* in T_2. The

graph cannot be conflict serializable and a schedule cannot be made as there are conflicting operations.

Schedule 2 is conflict serializable. The following graph shows there are no cycles in the graph.



There are no cycles and we can thus obtain a schedule. The schedule is presented below:

T_1	T_2	T_3	T_4	T_5
read(X)				
read(Y)				
read(Z)				
	write(X)			
	write(Y)			
		read(X)		
		write(Z)		
			read(Z)	
			write(Z)	
				write(Y)