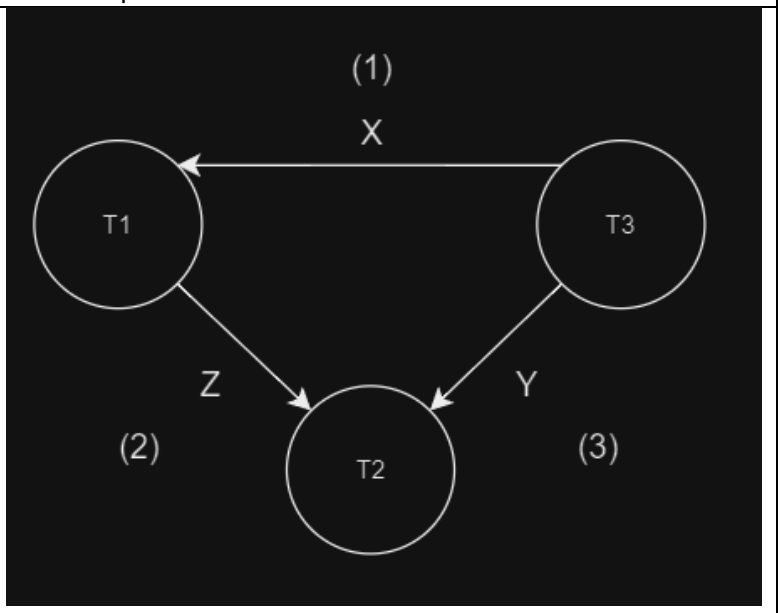
Parcialito 5

Alumno: Alan Ezequiel Valdevenito.
Padrón: 107585.

Ejercicio 1

Item a

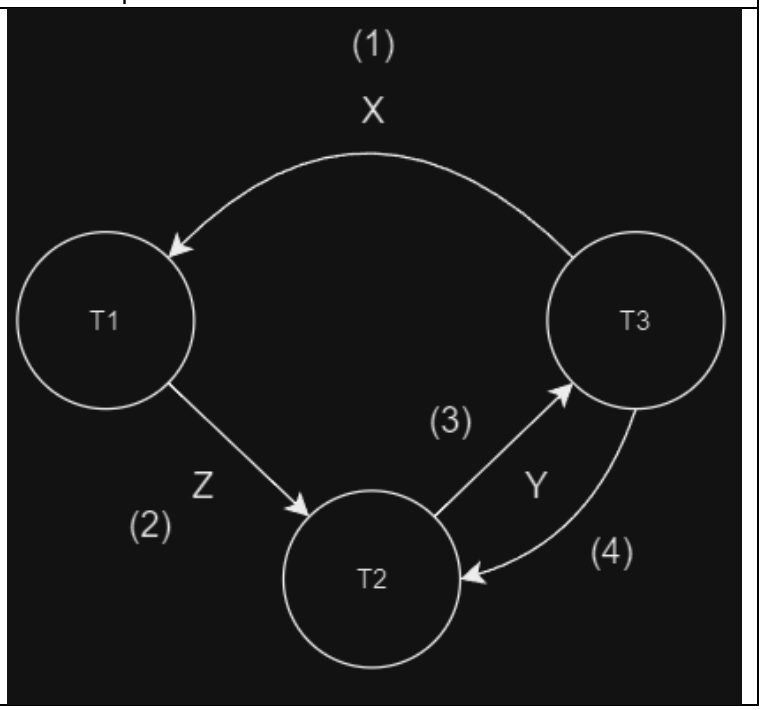
T1	T2	T3	Grafo de precedencias
bT1 RT1(X) **RT1(Z)** **WT1(X)** cT1	bT2 RT2(Z) **RT2(Y)** **WT2(Z)** **WT2(Y)** cT2	bT3 **RT3(X)** **RT3(Y)** **WT3(Y)** cT3	

Conflictos:

1) T3 tiene que preceder a T1 ya que T3 lee X y luego T1 modifica X (sección roja).
2) T1 tiene que preceder a T2 ya que T1 lee Z y luego T2 modifica Z (sección verde).
3) T3 tiene que preceder a T2 ya que T3 lee/modifica Y y luego T2 lee/modifica Y (sección azul).

Un orden de ejecución es serializable por conflictos si y sólo si su grafo de precedencias no tiene ciclos. Luego, como no tenemos un ciclo podemos afirmar que el solapamiento es serializable.

Item b

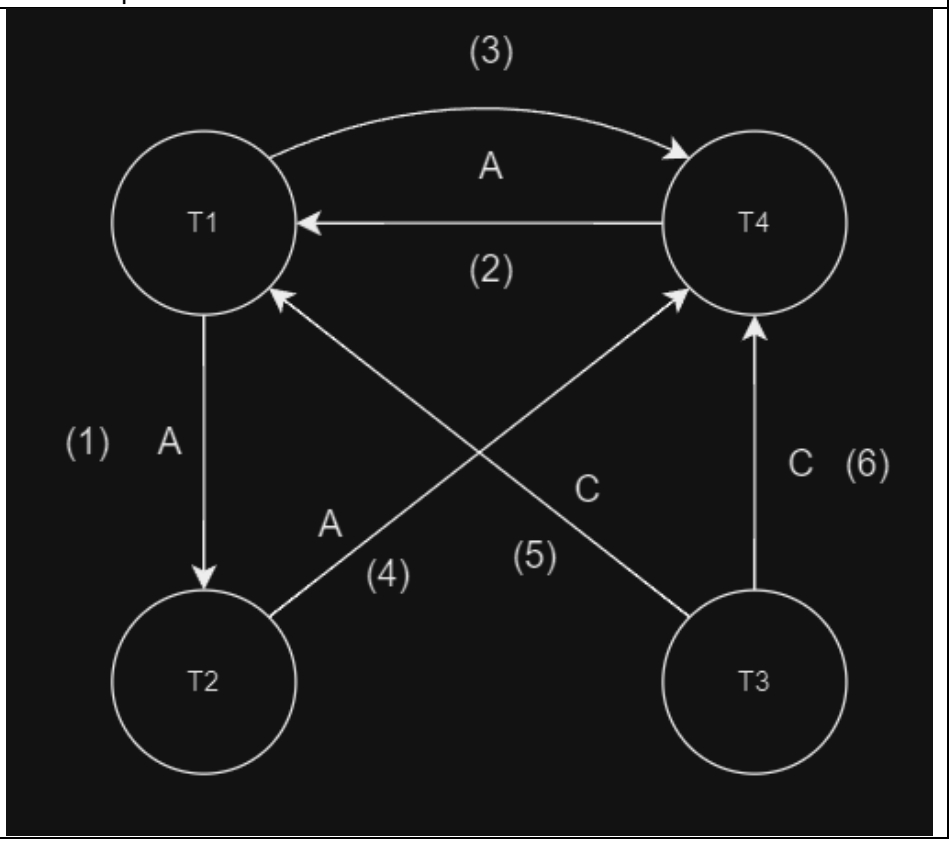
T1	T2	T3	Grafo de precedencias
bT1 RT1(X) **RT1(Z)** **WT1(X)** cT1	bT2 RT2(Z) **RT2(Y)** **WT2(Z)** **WT2(Y)** cT2	bT3 **RT3(X)** **RT3(Y)** **WT3(Y)** cT3	

Conflictos:

1) T3 tiene que preceder a T1 ya que T3 lee X y luego T1 modifica X (sección roja).
2) T1 tiene que preceder a T2 ya que T1 lee Z y luego T2 modifica Z (sección verde).
3) T2 tiene que preceder a T3 ya que T2 lee Y y luego T3 modifica Y (sección azul).
4) T3 tiene que preceder a T2 ya que T3 lee Y y luego T2 modifica Y (sección rosa).

Un orden de ejecución es serializable por conflictos si y sólo si su grafo de precedencias no tiene ciclos. Luego, como tenemos al menos un ciclo podemos afirmar que el solapamiento no es serializable.

Item c

T1	T2	T3	T4	Grafo de precedencias
bT1 **RT1(A)** **WT1(A)** **RT1(C)** cT1	bT2 RT2(B) **RT2(A)** WT2(B) cT2	bT3 **RT3(C)** **WT3(C)** cT3	bT4 **RT4(A)** **RT4(C)** **WT4(A)** **WT4(C)** cT4	

Conflictos:

1) T1 tiene que preceder a T2 ya que T1 modifica A y luego T2 lee A (sección roja).
2) T4 tiene que preceder a T1 ya que T4 lee A y luego T1 modifica A (sección roja).
3) T1 tiene que preceder a T4 ya que T1 lee/modifica A y luego T4 modifica A (sección roja).
4) T2 tiene que preceder a T4 ya que T2 lee A y luego T4 modifica A (sección roja).
5) T3 tiene que preceder a T1 ya que T3 modifica C y luego T1 lee C (sección azul).
6) T3 tiene que preceder a T4 ya que T3 lee/modifica C y luego T4 lee/modifica C (sección azul)

Un orden de ejecución es serializable por conflictos si y sólo si su grafo de precedencias no tiene ciclos. Luego, como tenemos al menos un ciclo podemos afirmar que el solapamiento no es serializable.

Ejercicio 2

Item a

2PL básico:

- El cumplimiento de este protocolo garantiza que cualquier orden de ejecución sea serializable.
- Una transacción T_i no puede adquirir un lock sobre un elemento que ya había lockeado.
- Las transacciones se dividen en dos fases de ejecución: adquisición de locks y liberación de locks.

Colocamos los locks y unlocks suponiendo un solapamiento arbitrario:

T1	T2
bT1 **LOCK(C)** RT1(C) WT1(C) **UNLOCK(C)** **LOCK(B)** RT1(B) WT1(B) **UNLOCK(B)** cT1	 bT2 **LOCK(A)** RT2(A) WT2(A) **UNLOCK(A)** **LOCK(C)** RT2(C) WT2(C) **UNLOCK(C)** cT2

Propongamos otro solapamiento donde se vea que si T1 hace LOCK(C) y T2 quiere también hacer LOCK(C) este pedido será denegado y deberá esperar a que T1 haga UNLOCK(C):

T1	T2
bT1 **LOCK(C)** RT1(C) WT1(C) **UNLOCK(C)** **LOCK(B)** RT1(B) WT1(B) **UNLOCK(B)** cT1	 bT2 **LOCK(A)** RT2(A) WT2(A) **UNLOCK(A)** **LOCK(C) DENEGADO** **LOCK(C)** RT2(C) WT2(C) **UNLOCK(C)** cT2

Como vemos, el resultado es el mismo que una ejecución serial.

Item b

Un solapamiento es recuperable si y sólo si ninguna transacción T realiza el commit hasta tanto todas las transacciones que escribieron datos antes de que T los leyera hayan commiteado.

Básicamente si tenemos una transacción T_i que lee/escribe datos en X y tenemos otra transacción T_j que también lee/escribe datos en X luego de T_i, para que el solapamiento sea recuperable T_i debe commitear antes que T_j.

Luego, es recuperable ya que si bien T1 lee y escribe datos en C y luego T2 también lee los datos y escribe en C, el commit lo realiza primero T1 y luego T2.

El solapamiento no seria recuperable si el commit lo hubiese realizado primero T2 y luego T1.

Ejercicio 3

Item a

Nro linea	log
1	<START T6>
2	<T6, X, 20, 80>
3	<START T7>
4	<T6, Y, 20, 150>
5	<T6, X, 80, 90>
6	<T7, Z, 30, 50>
7	<T7, W, 40, 70>
8	<COMMIT T6>
9	<START T8>
10	<T8, U, 50, 120>
11	<T7, W, 70, 100>
12	<START CKPT(T7,T8)>
13	<T7, Z, 50, 110>
14	<COMMIT T7>
15	<START T9>
16	<T9, V, 60, 140>
17	<T9, T, 70, 20>
18	<COMMIT T8>

Comenzamos el procedimiento de recuperación.

Situación: Encontramos primero un registro (BEGIN CKPT) en la línea 12.

¿Qué transacciones hicieron COMMIT?: T6, T7 y T8.
¿Qué transacciones no hicieron COMMIT?: T9.

Aplicamos la recuperación del algoritmo UNDO/REDO:

1) Recorremos el log completo de adelante hacia atrás y por cada transacción que no hizo COMMIT aplicamos cada uno de los WRITE para restaurar el valor anterior a la misma en disco (UNDO):

$$Deshacemos\ T9: T \rightarrow 70$$
$$Deshacemos\ T9: V \rightarrow 60$$

2) Recorremos el log de atrás hacia adelante volviendo a aplicar cada uno de los WRITE de las transacciones que commitearon, para asegurar que quede asignado el nuevo valor de cada ítem (REDO):

$$Rehacemos\ T6: X \rightarrow 80$$
$$Rehacemos\ T6: Y \rightarrow 150$$
$$Rehacemos\ T6: X \rightarrow 90$$
$$Rehacemos\ T7: Z \rightarrow 50$$
$$Rehacemos\ T7: W \rightarrow 70$$
$$Rehacemos\ T8: U \rightarrow 120$$
$$Rehacemos\ T7: W \rightarrow 100$$
$$Rehacemos\ T7: Z \rightarrow 110$$

3) Escribimos $(ABORT; T_9)$ en el log para que el mismo quede consistente y hacemos flush del log a disco.

Por lo tanto, el valor de los items es:

- X = 90
- Y = 150
- Z = 110
- W = 100
- U = 120
- V = 60
- T = 70

Item b

Nro linea	log
1	<START T6>
2	<T6, X, 20, 80>
3	<START T7>
4	<T6, Y, 20, 150>
5	<T6, X, 80, 90>
6	<T7, Z, 30, 50>
7	<T7, W, 40, 70>
8	<COMMIT T6>
9	<START T8>
10	<T8, U, 50, 120>
11	<T7, W, 70, 100>
12	<START CKPT(T7,T8)>
13	<T7, Z, 50, 110>
14	<COMMIT T7>
15	<START T9>
16	<T9, V, 60, 140>
17	<T9, T, 70, 20>
18	<COMMIT T8>
19	<T9, V, 140, 200>
20	<START T10>
21	<T10, Z, 110, 180>
22	<END CKPT>
23	<T9, V, 200, 170>

Comenzamos el procedimiento de recuperación.

Situación: Encontramos primero un registro (END CKPT) en la línea 22.

El checkpoint asegura que los cambios realizados por T6 están en disco por haber hecho COMMIT antes del registro (BEGIN CKPT). Asimismo, los cambios realizados por T7 y T8 antes del registro (BEGIN CKPT) también están en disco.

¿Qué transacciones hicieron COMMIT?: T6, T7 y T8.
¿Qué transacciones no hicieron COMMIT?: T9 y T10.

Aplicamos la recuperación del algoritmo UNDO/REDO:

1) Recorremos el log completo de adelante hacia atrás y por cada transacción que no hizo COMMIT aplicamos cada uno de los WRITE para restaurar el valor anterior a la misma en disco (UNDO):

$$Deshacemos\ T9: V \rightarrow 200$$
$$Deshacemos\ T10: Z \rightarrow 110$$
$$Deshacemos\ T9: V \rightarrow 140$$
$$Deshacemos\ T9: T \rightarrow 70$$
$$Deshacemos\ T9: V \rightarrow 60$$

2) Recorremos el log de atrás hacia adelante desde el registro (BEGIN CKPT) volviendo a aplicar cada uno de los WRITE de las transacciones que commitearon, para asegurar que quede asignado el nuevo valor de cada ítem (REDO):

$$Rehacemos\ T7: Z \rightarrow 110$$

3) Escribimos $(ABORT; T_9)$ y $(ABORT; T_{10})$ en el log para que el mismo quede consistente y hacemos flush del log a disco.

Por lo tanto, el valor de los items es:

- X = 90
- Y = 150
- Z = 110
- W = 100
- U = 120
- V = 60
- T = 70

Item c

Nro linea	log
1	<START T6>
2	<T6, X, 20, 80>
3	<START T7>
4	<T6, Y, 20, 150>
5	<T6, X, 80, 90>
6	<T7, Z, 30, 50>
7	<T7, W, 40, 70>
8	<COMMIT T6>
9	<START T8>
10	<T8, U, 50, 120>
11	<T7, W, 70, 100>
12	<START CKPT(T7,T8)>
13	<T7, Z, 50, 110>
14	<COMMIT T7>
15	<START T9>
16	<T9, V, 60, 140>
17	<T9, T, 70, 20>
18	<COMMIT T8>
19	<T9, V, 140, 200>
20	<START T10>
21	<T10, Z, 110, 180>
22	<END CKPT>
23	<T9, V, 200, 170>
24	<COMMIT T9>

Comenzamos el procedimiento de recuperación.

Situación: Encontramos primero un registro (END CKPT) en la línea 22.

El checkpoint asegura que los cambios realizados por T6 están en disco por haber hecho COMMIT antes del registro (BEGIN CKPT). Asimismo, los cambios realizados por T7 y T8 antes del registro (BEGIN CKPT) también están en disco.

¿Qué transacciones hicieron COMMIT?: T6, T7, T8 y T9.
¿Qué transacciones no hicieron COMMIT?: T10.

Aplicamos la recuperación del algoritmo UNDO/REDO:

1) Recorremos el log completo de adelante hacia atrás y por cada transacción que no hizo COMMIT aplicamos cada uno de los WRITE para restaurar el valor anterior a la misma en disco (UNDO):

$$Deshacemos\ T10: Z \rightarrow 110$$

2) Recorremos el log de atrás hacia adelante desde el registro (BEGIN CKPT) volviendo a aplicar cada uno de los WRITE de las transacciones que commitearon, para asegurar que quede asignado el nuevo valor de cada ítem (REDO):

$$Rehacemos\ T7: Z \rightarrow 110$$
$$Rehacemos\ T9: V \rightarrow 140$$
$$Rehacemos\ T9: T \rightarrow 20$$
$$Rehacemos\ T9: V \rightarrow 200$$
$$Rehacemos\ T9: V \rightarrow 170$$

3) Escribimos $(ABORT; T_{10})$ en el log para que el mismo quede consistente y hacemos flush del log a disco.

Por lo tanto, el valor de los items es:

- X = 90
- Y = 150
- Z = 110
- W = 100
- U = 120
- V = 170
- T = 20