UNIVERSIDAD DE BUENOS AIRES

FACULTAD DE CIENCIAS EXACTAS Y NATURALES

DEPARTAMENTO DE COMPUTACIÓN

BASES DE DATOS
CONCURRENCIA Y RECUPERABILIDAD

GUÍA DE EJERCICIOS

---

1 Serializabilidad / Recuperabilidad de Historias

1.1. Dada la siguiente historia (*schedule*) que es *conflicto-serializable* transformarla en una historia serial mediante una secuencia de intercambios no conflictivos de acciones adyacentes.

$H_1 = r_1(A); w_1(A); r_2(A); w_2(A); r_1(B); w_1(B); r_2(B); w_2(B);$

1.2. Dadas las siguientes historias (*schedules*):

$H_1 = \ r_1(X);\ r_4(Y);\ w_1(X);\ w_4(X);\ r_3(X);\ c_1;\ w_4(Y);\ w_3(Y);\ w_4(Z);\ c_4;\ w_3(X);\ c_3.$
$H_2 = \ r_1(X);\ w_2(X);\ r_1(Y);\ w_1(X);\ w_1(Y);\ c_1;\ r_2(Z);\ w_2(Y);\ c_2.$
$H_3 = \ w_1(X);\ w_2(X);\ w2_(Y);\ c_2;\ w_1(Y);\ c_1;\ w_3(X);\ w_3(Y);\ c_3.$
$H_4 = \ w_2(X);\ r_1(X);\ r_2(Z);\ r_1(Y);\ w_2(Y);\ w_1(Y);\ c_2;\ w_1(X);\ c_1.$
$H_5 = \ r_1(X);\ r_4(Y);\ r_3(X);\ w_1(X);\ w_4(Y);\ w_3(X);\ c_1;\ w_3(Y);\ w_4(Z);\ c_3;\ W_4(X);\ c_4.$

(a) Indicar para cada una si es NoRC, RC, ACA o ST.

(b) Indicar cuáles podrían producir un *dirty read* o un *lost update*.

1.3. Clasificar según recuperabilidad las siguientes historias:

$H_1 = r_1(X);\ r_2(Z);\ r_1(Z);\ r_3(X);\ r_3(Y);\ w1(X);\ c_1;\ w_3(Y);\ c_3;\ r_2(Y);\ w_2(Z);\ w_2(Y);\ c_2.$
$H2 = r_1(X);\ r_2(Z);\ r_1(Z);\ r_3(X);\ r_3(Y);\ w1(X);\ w_3(Y);\ r_2(Y);\ w_2(Z);\ w_2(Y);\ c_1;\ c_2;\ c_3.$
$H3 = r_1(X);\ r_2(Z);\ r_3(X);\ r_1(Z);\ r_2(Y);\ r_3(Y);\ w1(X);\ c_1;\ w_2(Z);\ w_3(Y);\ w_2(Y);\ c_3;\ c_2.$

1.4. Dadas las siguientes transacciones:

$T_1 = w_1(B);\ r_1(C);\ w(C);$
$T_2 = r_2(D)\ r_2(B);\ w_2(C);$

(a) Dar una historia H_1 que no sea RC.

(b) Dar una historia H_2 que sea ACA pero no ST.

(c) Dar una historia H_3 que sea ST.

2 Introducción a Protocolos de Bloqueo o *Locking*

2.1. Dada las siguientes transacciones escritas como Read/Write:

$T_1 = r_1(A);\ w_1(A);\ r_1(B);\ w_1(B)$
$T_2 = r_2(A);\ w_2(A);\ r_2(B);\ w_2(B)$

Nota: Si bien ambas parecen iguales en realidad pueden realizar operaciones diferentes sobre los ítems leídos.

(a) Incorporar locks para que llevarlas a un modelo con *locking* binario

(b) Realizar una historia legal pero no serializable

2.2. Dadas las siguientes transacciones que deben ser ejecutadas en forma concurrente, indicar una historia legal en el cual se produzca un dirty read:

$T_1 = l_1(A); A = A + 2; l_1(B); B = A + 5; u_1(B); u_1(A); l_1(C); C = 2 * C; u_1(C)$

$T2 = l_2(A); A = A + 1; l_2(E); E = A + 8; l_2(D); u_2(E); D = D/5; u_2(A); u_2(D)$

2.3. Considerando la siguiente historia con el modelo introductorio de LOCK/UNLOCK.

(a) Determinar si es serializable.

(b) En caso afirmativo, proponer un schedule serial equivalente

T_1	T_2	T_3	T_4
$l(A)$			
	$l(C)$		
$l(B)$			
	$u(C)$		
$u(A)$			
$u(B)$			
	$l(B)$		
	$u(B)$		
		$l(B)$	
		$u(B)$	
			$l(C)$
			$l(D)$
			$l(E)$
			$u(C)$
		$l(C)$	
			$u(D)$
		$u(C)$	
			$u(E)$

2.4. Dadas las transacciones del ejercicio 2.1 llevarlas a un modelo de locking binario tal que ambas sean 2PL.

2.5. Dadas las siguientes transacciones:

$T_1 = l_1(A); l_1(B); u_1(A); u_1(B)$

$T_2 = l_2(B); l_2(A); u_2(B); u_2(A)$

(a) Responder: ¿Cumplen las transacciones T_1 y T_2 con 2PL?

(b) Realizar un entrelazado tal que se produzca un *deadlock*

3 Sistemas de Bloqueo con varios modos

3.1. Dadas las siguientes transacciones T_1 y T_2.

$T_1 = wl_1(A); A = A + 1; wl_1(B); B = A + B; u_1(A); u_1(B)$

$T_2 = rl_2(A); wl_2(C); C = A + 1; wl_2(D); D = 1; u_2(A); u_2(D); u_2(C)$

(a) Construir un schedule legal serializable que no sea serial

(b) Responder: ¿Cumplen las transacciones T_1 y T_2 con 2PL?

3.2. Dadas las transacciones T_1 , T_2 , T_3 y T_4 con el siguiente entrelazado:

T_1	T_2	T_3	T_4
	$rl(A)$		
		$rl(A)$	
	$wl(B)$		
	$u(A)$		
		$wl(A)$	
	$u(B)$		
$rl(B)$			
		$u(A)$	
			$rl(B)$
$rl(A)$			
			$u(B)$
$wl(C)$			
$u(A)$			
			$wl(A)$
			$u(A)$
$u(B)$			
$u(C)$			

(a) Determinar si es serializable y, en caso afirmativo, proponer las correspondientes historias seriales equivalentes.

3.3. Dadas las transacciones T_1 , T_2 , T_3 , T_4 y T_5 con el siguiente entrelazado:

$H_1 : rl_1(A); rl_2(A); rl_3(A); u_1(A); rl_5(B); u_5(B), wl_1(B); wl_2(C); u_1(B); rl_3(B); u_3(A); u_2(A);$
$wl_4(A); u_4(A); rl_1(D); u_1(D); wl_4(D); u_4(D); wl_5(A); u_5(A); u_2(C); u_3(B)$

(a) Dibujar el entrelazado en forma tabular.

(b) Determinar si es serializable y, en caso afirmativo, proponer las correspondientes historias seriales equivalentes.

(c) Para cada transacción indicar si cumple con ser 2PL. Si no lo son modificarlas para que lo sean.

(d) Sobre el ítem anterior ubicar *commits* para que sean 2PL estricto y 2PL riguroso.

3.4. Dadas las transacciones:

$T_1 : rl_1(A); wl_1(B); u_1(A); u_1(B)$
$T_2 : rl_2(A); u_2(A); rl_2(B); u_2(B)$
$T_3 : wl_3(A); u_3(A); wl_3(B); u_3(B)$
$T_4 : rl_4(B); u_4(B); wl_4(A); u_4(A)$

(a) Armar una historia legal y serializable y dar la correspondiente planificación serial.

(b) Dibujar el Grafo de Precedencia de la siguiente historia H_1 ($SG(H_1)$) y determinar si es serializable:
$H_1 : wl_3(A); rl_4(B); u_3(A); rl_1(A); u_4(B); wl_3(B); rl_2(A); u_3(B); wl_1(B); u_2(A);$
$u_1(A); wl_4(A); u_1(B); rl_2(B); u_4(A); u_2(B)$

(c) Reescribir las transacciones para que adhieran al protocolo 2PL.

3.5. Dadas las transacciones T_1 , T_2 , T_3 con el siguiente entrelazado H_1:

$H_1 : wl_3(B); rl_3(B); rl_1(A); rl_3(C); u_3(B); rl_1(B); u_1(B); wl_2(B) : u_3(C); rl_2(C);$
$u_3(A); u_1(A); wl_2(A); u_2(B); u_2(A); rl_1(C); rl_3(D); u_3(D); u_1(C); wl_2(C); u_2(C)$

(a) Indicar a qué historia serial es equivalente H_1. Justificar la respuesta.

(b) Responder: ¿Existe algún schedule legal no serializable que se pueda construir con T1, T2 y T3? Justificar la respuesta. En caso afirmativo, mostrar un ejemplo y justificar.

(c) Responder: ¿Es posible identificar en H_1 un caso en el que pueda producirse un *dirty read*? Justificar la respuesta. En caso afirmativo, mostrar un ejemplo y justificar

Nota: Dibujar el entrelazado en forma tabular para poder visualizar mejor la historia.

3.6. Dada la siguiente historia H_1, sobre el conjunto de transacciones T_1, T_2, T_3, T_4 y el conjunto de items X, Y:

$H_1 = rl_3(X); rl_2(X); wl_3(Y); u_3(X); wl_2(X); u_3(Y); rl_4(Y); u_2(X); rl_1(Y); rl_4(X); u_1(Y); u_4(X); wl_1(X); u_4(Y); u_1(X); c_3; c_2; c_4; c_1$

Nota: Asuma que entre un wl y su correspondiente ul ocurre solamente una escritura del ítem (no ocurre una lectura)

(a) Responder: ¿H_1 es Legal? ¿Todas la T_i son 2PL? .

(b) Hacer el $SG(H_1)$ e indicar si es serializable. En caso afirmativo obtener todas las historias seriales equivalentes.

(c) Clasificar H_1 con respecto a recuperabilidad: ST, ACA, RC, (no RC). Justificar

(d) Idem anterior pero cambiando el orden de los commits por: $c_4; c_1; c_2; c_3$

(e) Ubicar los commits y modificar H_1 para que las transacciones sean 2PL Estrictas

3.7. Dadas las siguientes transacciones:

$T_1 = rl_1(B); u_1(B); wl_1(A); u_1(A)$
$T_2 = rl_2(A); wl_2(A); u_2(A)$
$T_3 = rl_3(A); rl_3(B); u_3(A); u_3(B)$

Nota: Asumir que entre un wl y su correspondiente ul ocurre solamente una escritura del ítem (no ocurre una lectura)

(a) Se pide hallar una historia H tal que sea: Legal; No Serial; Serializable; Equivalente a la ejecución serial T_3, T_2, T_1 y ACA.

(b) Hacer el $SG(H)$ para verificar la seriabilidad del H hallado. Justificar las respuestas con respecto a legalidad y recuperabilidad.

(c) Transformar las transacciones a 2PL Estricto y 2PL Riguroso

3.8. Dadas las siguientes transacciones:

$T_1 = rl_1(A); r_1(A); rl_1(B); r_1(B); wl_1(B); w_1(B); u_1(A); u_1(B)$
$T_2 = rl_2(A); r_2(A); rl_2(B); r_2(B); u_2(A); u_2(B)$

(a) Realizar un entrelazado serializable no serial

(b) Proponer un entrelazado donde un *upgrade lock* sea denegado y deba esperar.

3.9. Tomando las siguientes transacciones

$T_1 = rl_1(A); r_1(A); wl_1(A); w_1(A); u_1(A)$
$T_2 = rl_2(A); r_2(A); wl_2(A); w_2(A); u_2(A)$

(a) Proponer un entrelazamiento que de un *deadlock* como resultado.

(b) Reescribir las transacciones, pero ahora, usando un modelo que soporte *update lock* y verificar si puede resolverse el *deadlock* anterior.

3.10. Dadas las siguientes historias

$H_1 = r_1(A); r_2(B); r_3(C); r_1(B); r_2(C); r_3(D); w_1(C); w_2(D); w_3(E)$
$H_2 = r_1(A); r_2(B); r_3(C); r_1(B); r_2(C); r_3(D); w_1(A); w_2(B); w_3(C)$
$H_3 = r_1(A); r_2(B); r_3(C); r_1(B); r_2(C); r_3(A); w_1(A); w_2(B); w_3(C)$

Se pide:

(a) Insertar locks de lectura y escritura de tal forma que **no haya** *upgrade locks*. Explicar que ocurriría en la ejecución con un planificador/scheduler que soporte locks de lectura y de escritura

(b) Insertar locks de lectura y escritura de tal forma que **si haya** *upgrade locks* y describir que ocurre con la ejecución.

(c) Insertar ahora locks de lectura, escritura y de actualización (*update locks*). Describir que ocurre en la ejecución.

3.11. Dadas las siguientes transaccioines

$T_1 = l_1(A); r_1(A); l_1(B); w_1(B); u_1(A); u_1(B)$
$T_2 = l_2(C); r_2(C); l_2(A); w_2(A); u_2(C); u_2(C)$
$T_3 = l_3(B); r_3(B); l_3(C); w_3(CA); u_3(B); u_3(C)$
$T_4 = l_4(D); r_4(D); l_4(A); w_4(A); u_4(D); u_4(A)$
Se pide:

(a) Transformar las mismas a un modelo de locks ternario

(b) Realizar un entrelazado tal que se produzca un deadlock y todas las transacciones queden esperando

(c) Realizar el grafo *wait-for*

(d) Describir que ocurre en el esquema anterior si se utiliza prevención de *deadlocks* con

i. Un esquema Wait-Die
ii. Un esquema Wound-Wait

3.12. Para cada una de las historias siguientes:

$H_1 = r_1(A); r_2(B); w_1(C); r_3(D); r_4(E); w_3(B); w_2(C); w_4(A); w_1(D);$
$H_2 = r_1(A); r_2(B); r_3(C); w_1(B); w_2(C); w_3(D);$
$H_3 = r_1(A); r_2(B); r_3(C); w_1(B); w_2(C); w_3(A);$
Se pide:

(a) Ubicar un *read lock* inmediatamente antes de cada acción de lectura, un *write lock* inmediatamente antes de cada acción de escritura y *unlocks* de modo que las transacciones cumplan con 2PL

(b) En base a lo anterior decir que acciones son denegadas y cuando ocurre un *deadlock*

(c) Dibujar la evolución del grafo *wait-for*

(d) Asumiendo un orden T_1, T_2, T_3, T_4 y un mecanismo de detección de deadlock explicar que ocuree si este mecanismo usa:

i. Un esquema Wait-Die
ii. Un esquema Wound-Wait

4 Métodos con *timestamping* y Multiversión

4.1. Detallar que ocurre en las siguientes secuencias, el orden es el orden temporal en el que ocurren los eventos si:

- $H_1 = st_1; st_2; r_1(A); r_2(B); w_2(A); w_1(B)$
- $H_2 = st_1; r_1(A); st_2; r_2(B); r_2(A); w_1(B)$
- $H_3 = st_1; st_2; st_3; r_1(A); r_2(B); w_1(C); r_3(B); r_3(C); w_2(B); w_3(B)$

(a) El planificador usa *timestamping* y no es multiversión.

(b) El planificador usa multiversión y protocolo **MVTO**

4.2. Dada la siguiente historia en un planificador con timestamp:

$H_1 = st_1; st_2; r_2(X); st_3; st_4; r_1(Y); r_4(Z); w_3(X); w_3(Y); w_4(Z); w_2(X); w_1(Y); r_3(Z)$

en donde los valores iniciales de $X; Y; Z$ son 0 y pasa que:

T_1 escribe $Y = 1$
T_2 escribe $X = 2$
T_3 escribe $X = 3; Y = 30$
T_4 escribe $Z = 4$

(a) Decir que pasa en cada acción y que valores quedan en X;Y;Z si el planificador ***no usa*** *multiversión*

(b) Decir que pasa en cada acción y que valores se tendrían para $X; Y; Z$ si el planificador **usa** *multiversión* **MVTO**

(c) Asumiendo un planificador multiversión, realizar el grafo de conflictos multiversión (GCM) e indicar si la historia **MCSR**

4.3. Dadas las siguientes transacciones:

$T_1 = r_1(A); r_1(B); w_1(X); w_1(B)$
$T_2 = r_2(C); r_2(B); r_2(A); w_2(A); w_2(B)$
$T_3 = r_3(B); r_3(X); w_3(A); w_3(B)$

Se pide:

(a) Realizar una historia para un planificador basado en timestamping multiversión **MVTO** que produzca un comportamiento físicamente irrealizable.

4.4. Sean T_1, T_2 y T_3, transacciones que operan sobre los mismos ítems A, B y C de una base de datos. Las siguientes secuencias muestran el orden de ejecución de las transacciones T_1, T_2 y T_3, .

(a) Asignar los timestamps e insertar los comienzos de transacción para que con timestamping sea posible realizar la secuencia sin que se produzca rollback.

(b) Modificar la segunda secuencia para que con multiversion **MV2PL** se produzca una espera en un paso **no final.**

Secuencias a analizar:

- $r_1(A); r_2(A); w_1(A); r_3(A); w_3(A); w_2(A)$
- $r_1(A); r_3(C); r_2(C); w_3(A); r_2(B); r_3(B); w_2(B); w_2(A)$

4.5. Dadas las siguientes historias:

$H_1 = st_2; r_2(Z); r_2(Y); w_2(Y); st_3; r_3(Y); r_3(Z); st_1; r_1(X); w_1(X); w_3(Y); w_3(Z); r_2(X); r_1(Y); w_1(Y); w_1(X)$
$H_2 = st_3; r_3(Y); r_3(Z); st_1; r_1(X); w_1(X); w_3(Y); w_3(Z); st_2; r_2(Z); r_1(Y); w_1(Y); r_2(Y); w_2(Y); r_2(X); w_2(X)$ Se pide:

(a) Verificar segun un planificador basado en timestamp si las historias son permitidas o no.

(b) Modificar una de las historias para adaptarla a un planificador multiversión **MV2PL** que demore alguna de las transacciones en **un paso final**.

(c) Modificar una de las historias para que un planificador multiversión **MVTO** tenga que hacer rollback.

4.6. Dadas las siguientes transacciones:

$T_1 = r_1(A); r_1(B); w_1(A)$
$T_2 = r_2(B); r_2(C); w_2(A)$
$T_3 = r_3(C); r_3(B); w_3(B)$

Se pide:

(a) Realizar una historia para un scheduler basado en *timestamp* **sin multiversión** tal que el orden inicio sea T_1, T_3, T_2 y que realice un rollback de T_3 por un comportamiento físicamente irrealizable *write too late*.

Nota: No olvidarse de indicar los comienzos de las transacciones.

5 Transacciones en SQL

5.1. Suponer que se crea una tabla con la siguiente sentencia:

CREATE TABLE Alumnos(idAlumno **INT NOT NULL PRIMARY KEY**,
nombre **VARCHAR**(90));

Luego se ejecutan las siguientes sentencias SQL:

INSERT INTO Alumnos (idAlumno, nombre) **VALUES** (1, 'Jhon␣Doe');
INSERT INTO Alumnos (idAlumno, nombre) **VALUES** (1, 'Don␣Nadie');
ROLLBACK;
SELECT $*$ **FROM** Alumnos;

Explicar que ocurre con la ejecución si:

(a) Están habilitadas las transacciones implícitas

(b) No están habilitadas las transacciones implícitas

5.2. En la misma estructura que el ejercicio anterior se ejecutan las siguientes sentencias SQL:

START **TRANSACTION**;
INSERT INTO Alumnos (idAlumno, nombre) **VALUES** (4, 'cuatro');
SELECT $*$ **FROM** Alumnos ;
ROLLBACK;
SELECT $*$ **FROM** Alumnos;

(a) Describir que ocurre y que devuelve cada sentencia

5.3. Considerar dos clientes diferentes del motor de base de datos trabajando concurrentemente. Se realiza la siguiente secuencia (el orden de arriba abajo es el orden temporal), la tabla Alumnos tiene previamente la tupla: (101, 'Horacio')

*/*Cliente 1*/*
START **TRANSACTION**;
SELECT NOMBRE **FROM** Alumnos **WHERE** idAlumno = 101 ;

*/*Cliente 2*/*
START **TRANSACTION**;
SELECT NOMBRE **FROM** Alumnos **WHERE** idAlumno = 101;
UPDATE Alumnos **SET** Nombre= 'Pepe' **WHERE** idAlumno = 101;
COMMIT;

*/*Cliente 1*/*
SELECT NOMBRE **FROM** Alumnos **WHERE** idAlumno = 101 ;
COMMIT;

Describir que ocurre con cada SELECT en el caso de que el nivel de aislamiento de las transacciones sea:

(a) REPEATABLE READ

(b) READ COMMITTED

5.4. Describir dos transacciones y un entrelazamiento entre ellas tal que el resultado sea diferente si se usa

(a) Aislamiento Serializable

(b) Aislamiento Snapshot

5.5. Dadas el siguiente código donde se realizan dos sesiones contra la base de datos en el orden en que están expuestas:

*/*Cliente 1*/*
BEGIN TRAN
UPDATE Person.Person
SET FirstName = 'James'
WHERE LastName = 'Jones';

*/*Cliente 2*/*
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SELECT FirstName, LastName
FROM Person.Person
WHERE LastName = 'Jones';

*/*Cliente 1*/*
ROLLBACK TRANSACTION;
SELECT FirstName, LastName
FROM Person.Person
WHERE LastName = 'Jones';

(a) Explicar que devolvería la consulta del *cliente 2* y la del *cliente 1*.

(b) Idem anterior pero si el nivel de aislamiento de la transacción del *cliente 2* fuera:

i. READ COMMITED
ii. REPEATABLE READ

5.6. Proponer un ejemplo de dos sesiones contra la misma base de datos tal que los niveles de aislamiento READ COMMITED y REPEATABLE READ den resultados diferentes.

6 Recuperabilidad

6.1. Dada la siguiente historia:

$H = r_1(Z); w_1(U); c_1; w_2(X); w_2(Y); r_3(U); w_3(X); w_2(Z); c_2; r_3(Y); r_4(Z); w_3(Y); c_3; r_4(U); w_4(U); c_4$

Suponer que inicialmente todos los items tienen valor 0 y cada operación WRITE incrementa en 1 el valor anterior. Escribir la secuencia de los registros de *log* correspondientes a la ejecución exitosa de H, según:

(a) *Undo logging* con *checkpointing quiescente* (incluir un *checkpoint* en el *log*)

(b) *Redo logging* con *checkpointing no quiescente* (incluir un *checkpoint* en el *log* tal que ocurra durante la ejecución de H)

(c) *Undo/Redo logging* sin *checkpointing*.

6.2. Dada la siguiente secuencia de registros en el *undo-log*, escritos por el *log manager*, correspondientes a la ejecución de tres transacciones T_1, T_2 y T_{12}, sobre los ítems A,B, C, D , E y R:

$< START\ T_1 >; < T_1, A, 10 >; < START\ T_2 >; < T_2, B, 20 >; < START\ T_{12} >$
$< T_1, C, 30 >; < T_2, D, 40 >; < COMMIT\ T_2 >; < T_12, R, 12 >; < T_1, E, 50 >;$
$< ABORT\ T12 >< COMMIT\ T1 >$

Describir las acciones que debería realizar el *recovery manager* (incluyendo los cambios tanto en los ítems en disco como en el *log*) si ocurre un *crash* y el último registro de *log* en disco es:

(a) $< START\ T_2 >$

(b) $< COMMIT\ T_2 >$

(c) $< T_1, E, 50 >$

(d) $< COMMIT\ T_1 >$

6.3. Dada la siguiente secuencia de registros en el *undo-log* generados por el log manager correspondientes a la ejecución de las transacciones T_1, T_2, T_3 y T_4:

$< START\ T_1 >; < T_1, A, 8 >; < START\ T_2 >; < START\ T_3 >< T_2, B, 16 >;$
$< START\ T_4 >; < T_4, E, 24 >; < T_2, D, 32 >; < T4, K, 9 >; < T_4, F, 40 >;$
$< ABORT\ T4 >; < COMMIT\ T_3 >; < T_2, G, 48 >; < COMMIT\ T_2 >; < T_1, C, 56 >;$
$< COMMIT\ T_1 >$

(a) Describir las acciones que debería realizar el *recovery manager* (incluyendo los cambios tanto en los ítems en disco como en el *log*) si ocurre un *crash* y el último registro de *log* en disco es:

i. $< START\ T_4 >$
ii. $< T_1, C, 56 >$

(b) Ídem ejercicio anterior pero asumiendo *redo logging* y que el último registro de *log* en disco es:

i. $< COMMIT\ T_3 >$
ii. $< COMMIT\ T_1 >$

6.4. Dada la siguiente secuencia de registros *undo/redo-log* generados por el *log manager* correspondientes a la ejecución de cuatro transacciones T_1, T_2, T_3 y T_4:

$< START\ T_1 >; < T_1, A, 100, 110 >; < START\ T_2 >; < T_2, B, 200, 210 >;$
$< START\ T3 >; < T_1, C, 300, 310 >; < T_3, D, 400, 410 >; < T_2, E, 40, 41 >;$
$< T_3, F, 500, 510 >; < COMMIT\ T_3 >; < COMMIT\ T_2 >; < START\ T_4 >;$
$< T_1, G, 600, 610 >; < T_4, H, 700, 710 >; < COMMIT\ T_1 >; < COMMIT\ T_4 >$

Describa las acciones que debería realizar el recovery manager -incluyendo los cambios tanto en los items en disco como en el log- si ocurre un crash y el último registro de log en disco es:

(a) $< START\ T_3 >$
(b) $< COMMIT\ T_3 >$
(c) $< T_1, G, 600, 610 >$
(d) $< COMMIT\ T_4 >$

6.5. Dada la siguiente secuencia de registros de *undo-log*:

$< STARTT_1 >; < T_1, A, 60 >; < COMMIT\ T_1 >; < START\ T_2 >; < T_2, A, 10 >;$
$< START\ T_3 >; < T_3, B, 20 >; < T_2, C, 30 >; < START\ T_4 >; < T_3, D, 40 >;$
$< T_4, F, 70 >; < COMMIT\ T_3 >; < T_2, E, 50 >; COMMIT\ T_2 >; < T_4, B, 80 >;$
$< COMMIT\ T_4 >$

Suponer que se inicia *checkpoint no-quiescente* inmediatamente después de que cada uno de los siguientes registros de *log* han sido escritos en el *pool* de *buffers* de memoria:

1) $< T_1, A, 60 >$
2) $< T_2, A, 10 >$
3) $< T_3, B, 20 >$
4) $< T_3, D, 40 >$
5) $< T_2, E, 50 >$

Para cada uno indicar:

(a) Cuándo el registro < END CKPT > es escrito.
(b) Para cada punto donde fuera posible que ocurriera un *crash*, hasta dónde se debería examinar el *log* para hallar todas las posibles transacciones incompletas.

6.6. Dados los mismos datos del ejercicio anterior pero ahora asumiendo ***redo logging*** indicar para cada punto de 1) a 5):

(a) En qué punto el registro < END CKPT > podría ser escrito.
(b) Para cada punto donde fuera posible que ocurriera un *crash*, hasta dónde se debería examinar el *log* para hallar todas las posibles transacciones completas. Considere los dos casos posibles
   i. que el registro < END CKPT > **fué** escrito antes del *crash*.
   ii. que el registro < END CKPT > **no fué** escrito antes del *crash*.

6.7. Dada la siguiente secuencia de registros *undo/redo-log* con *nonquiescent checkpointing* generados por el *log manager* correspondientes a la ejecución de las transacciones T_1 a T_5:

1. $<$ START $T_1 >$
2. $< T_1, A, 26, 33 >$
3. $<$ START $T_2 >$
4. $<$ START $T_3 >$
5. $< T_2, C, 51, 34 >$
6. $< T_3, E, 21, 35 >$
7. $< T_1, B, 25, 24 >$
8. $< T_1, A, 33, 20 >$
9. $< T_2, D, 18, 36 >$
10. $<$ COMMIT $T_1 >$
11. $<$ START $T_4 >$
12. $< T_4, F, 19, 37 >$
13. $< T_2, D, 36, 4 >$
14. $<$ START CKPT $(T_2, T_3, T_4) >$
15. $< T_2, C, 34, 10$¿
16. $< T_4, F, 37, 41 >$
17. $<$ START $T_5 >$
18. $< T_3, E, 35, 52 >$
19. $< T_5, G, 27, 43 >$
20. $<$ COMMIT $T_2 >$
21. $<$ END CKPT $>$
22. $< T_5, G, 43, 28 >$
23. $< T_4, H, 26, 23 >$
24. $<$ COMMIT $T_5 >$
25. $<$ COMMIT $T_4 >$

Asuma que las entradas en el log son de la forma $< transaccion, item, valor\ anterior, valor\ nuevo >$. Para los siguientes posibles escenarios de fallas:

(a) El sistema *crashea* justo antes de que la línea 23 sea escrita en el disco.

(b) Idem anterior pero para la línea 24.

(c) Idem anterior pero para la línea 25.

(d) El sistema *crashea* inmediatamente después de que la línea 25 fuera escrita en el disco.

Completar una tabla con los valores de los items A, B, C, D, E, F, G y H en el disco después de una recuperación exitosa.

7 Notación

En esta sección veremos un resumen de la notación utilizada para escribir las transacciones y las historias.

H_i	**Historia**. Historia o *schedule* i.
T_i	**Transacción**. Forma de notar a la transacción i.
$r_i(A)$	**Lectura**. La transacción i lee el ítem A.
$w_i(A)$	**Escritura**. La transacción i escribe el ítem A.
c_i	**Commit**. La transacción i realiza un *commit*.
a_i	**Abort**. Se aborta la transacción i .
$l_i(A)$	**Lock**. La transacción i realiza un *bloqueo o lock* sobre el ítem A. Usado en lockeo binario.
$u_i(A)$	**UnLock**. La transacción i libera los *bloqueos o locks* previos sobre el ítem A. Usado en todos los modelos, se asume que libera todos los *locks* tomados.
$rl_i(A)$	**Lock de lectura o compartido**. La transacción i realiza un *bloqueo o lock* de lectura sobre el ítem A.
$wl_i(A)$	**Lock de escritura o exclusivo**. La transacción i realiza un *bloqueo o lock* exclusivo o de escritura sobre el ítem A.
$ul_i(A)$	**Lock de actualización**. La transacción i realiza un *bloqueo o lock* de actualización (*update lock*)sobre el ítem A.
st_i	**Comienzo de transacción**. La transacción i comienza y se le asigna un *timestamp*. Usado en planificadores con *timestamp*.
$R_i(A, B, ...)$	**Comienzo y Conjunto de lectura**. La transacción i comienza con el conjunto de lectura $(A, B...)$. Usado para planificadores basados en validación.
$W_i(A, B, ...)$	**Finalización y Conjunto de escritura**. La transacción i finaliza con el conjunto de escritura $(A, B...)$. Usado para planificadores basados en validación.
V_i	**Validación**. La transacción i es validada. Usado para planificadores basados en validación.
$TS(T)$	**Timestamp** indica el timestamp de la transacción T