Administración de Transacciones

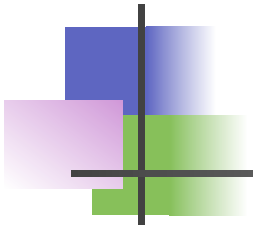
Planes Conflicto Equivalentes: Ejercicio

- Consideremos el siguiente plan (R=Lectura, W=Escritura, C=Commit, A=Abort)

$$S = R_3(c), W_2(a), W_2(b), R_1(a), R_3(a), R_2(c), R_3(b), C_3, W_1(a), C_2, C_1$$

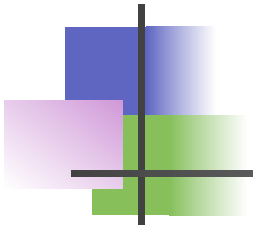
- ¿El plan es conflicto equivalente?
- ¿A qué plan es equivalente?

T1	T2	T3
		R(c)
	W(a)	
	W(b)	
R(a)		
		R(a)
	R(c)	
		R(b)
		commit
W(a)		
	commit	
commit		



Administración de Transacciones

- Grafos de Precedencia:
  - Es posible capturar los potenciales conflictos entre transacciones de un plan en un grafo de precedencia
  - El grafo de precedencia G(S) para un plan S se construye de la siguiente manera:
    - Cada transacción que compromete en S es un nodo en G(S)
    - Existe un arco desde T_i a T_j si una acción de T_i precede y es conflictiva con alguna acción de T_j
  - Un plan S es conflicto serializable sí y solo sí su grafo de precedencia es acíclico (no contiene ciclos)



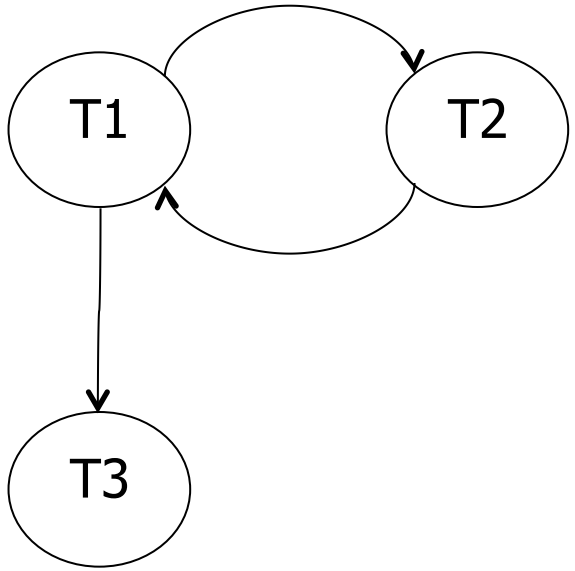
Administración de Transacciones

- Grafos de Precedencia:
  - Existe un arco desde T_i a T_j si una acción de T_i precede y es conflictiva con alguna acción de T_j, es decir se cumple una de las siguientes 3 condiciones:
    - T_i ejecuta write(Q) antes de que T_j ejecute read(Q)
    - T_i ejecuta read(Q) antes de que T_j ejecute write(Q)
    - T_i ejecuta write(Q) antes de que T_j ejecute write(Q)

Administración de Transacciones

- Grafos de Precedencia:
  - El siguiente plan no es conflicto equivalente porque existe un ciclo en G(S)

T_1	T_2	T_3
$R(A)$		
	$W(A)$	
	$commit_{T_2}$	
$W(A)$		
$commit_{T_1}$		
		$W(A)$
		$commit_{T_3}$



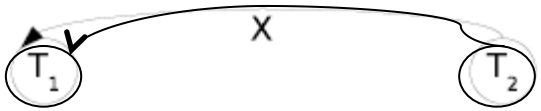
Administración de Transacciones

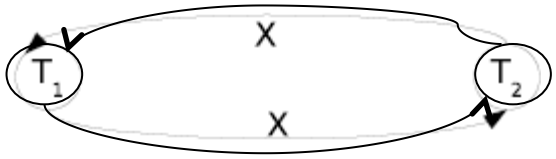
- Ejemplos Grafos de Precedencia:

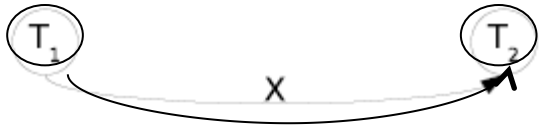
T_1	T_2
	read_item(X);
	$X := X + M$;
	write_item(X);
read_item(X);	
$X := X - N$;	
write_item(X);	
read_item(Y);	
$Y := Y + N$;	
write_item(Y);	

T_1	T_2
read_item(X);	
$X := X - N$;	
	read_item(X);
	$X := X + M$;
write_item(X);	
read_item(Y);	
	write_item(X);
$Y := Y + N$;	
write_item(Y);	

T_1	T_2
read_item(X);	
$X := X - N$;	
write_item(X);	
	read_item(X);
	$X := X + M$;
	write_item(X);
read_item(Y);	
$Y := Y + N$;	
write_item(Y);	



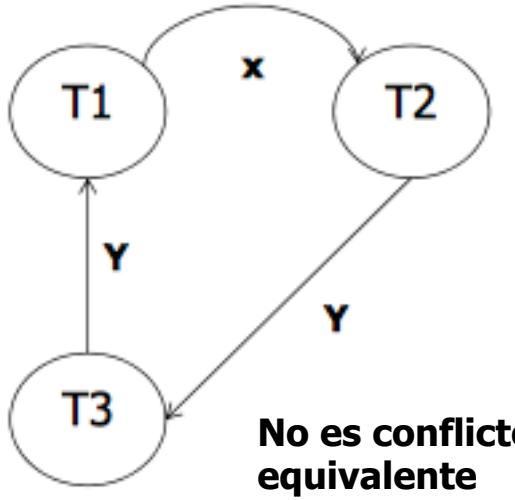




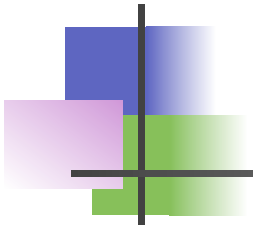
Administración de Transacciones

- Ejercicio Grafos de Precedencia:

T_1	T_2	T_3
	read_item(Z)	
	read_item(Y)	
	write_item(Y)	
		read_item(Y)
		read_item(Z)
read_item(X)		
write_item(X)		
		write_item(Y)
		write_item(Z)
	read_item(X)	
read_item(Y)		
write_item(Y)		
	write_item(X)	

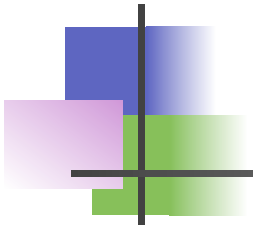


No es conflicto equivalente



Administración de Transacciones

- Control de Concurrencia Basado en Bloqueos
  - Un SGBD solo debe permitir la ejecución de planes que son serializables y recuperables
  - Para ello, el SGBD utiliza un protocolo de bloqueos
  - Tenemos dos tipos de bloqueos (candados):
    - Exclusivos X
    - Compartidos S (shared)
  - El protocolo de bloqueo más usado se denomina bloqueo exclusivo de dos fases (2PL exclusivo)

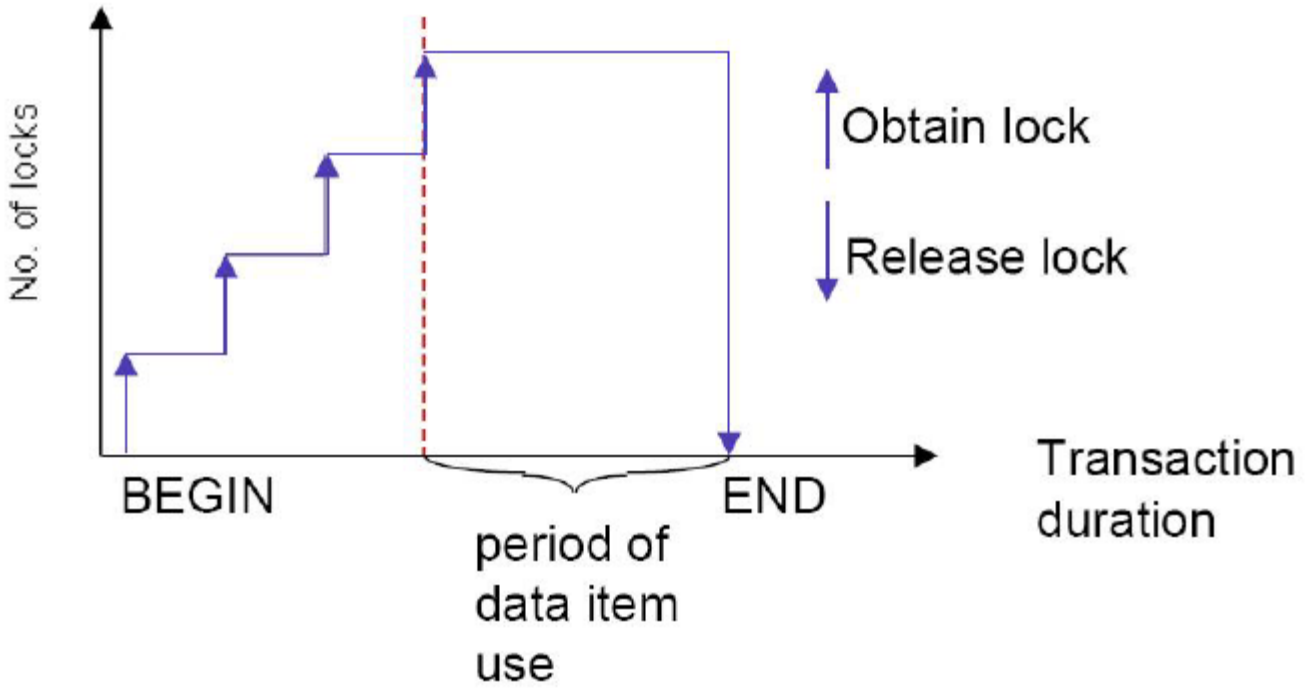


Administración de Transacciones

- Bloqueo Exclusivo de Dos Fases
  - El protocolo 2PL exclusivo (estricto) tiene dos reglas:
    1. Si una transacción T quiere leer (respectivamente, modificar) un objeto, primero solicita un bloqueo compartido (respectivamente, exclusivo) sobre el objeto
    2. Todos los bloqueos concedidos a una transacción se liberan cuando la transacción se completa
  - Obviamente, si una transacción tiene un bloqueo exclusivo sobre un objeto, también puede leer el objeto
  - Una transacción que requiere un bloqueo es suspendida hasta que el SGBD pueda otorgar el candado

Administración de Transacciones

- Bloqueo Exclusivo de Dos Fases



Administración de Transacciones

- Bloqueo Exclusivo de Dos Fases
  - Si una transacción tiene un candado exclusivo sobre un objeto, no puede haber otra transacción con un candado exclusivo o compartido sobre el objeto
  - La siguiente tabla resume la entrega de candados:

	X	S
X	NO	NO
S	NO	SI

  - El protocolo 2PL estricto permite que solo los planes seguros sean ejecutados y asegura que el grafo de precedencia de cualquier plan permitido sea acíclico
  - S_T (O) denota la solicitud de un candado compartido por la transacción T sobre el objeto O (X_T (O), respectivamente para candado exclusivo)

Administración de Transacciones

- Ejemplo: Protocolo 2PL Estricto
  - Consideremos las transacciones T1 y T2:
    - T1 transfiere 100 desde la cuenta A a la B
    - T2 incrementa A y B por el 10%
  - Con valores iniciales de A = 1000 y B = 500
  - El siguiente plan no está permitido por el protocolo 2PL estricto:

T_1	T_2
$R(A)$, A=1000	
$W(A)$, A=900	
	$R(A)$, A=900
	$W(A)$, A=990
	$R(B)$, B=500
	$W(B)$, B=550
	$commit_{T_2}$
$R(B)$= 550	
$W(B)$= 650	
$commit_{T_1}$	

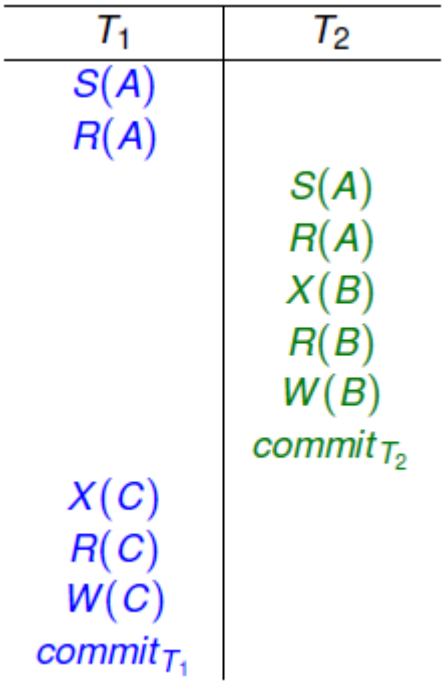
Administración de Transacciones

- Ejemplo: Protocolo 2PL Estricto
  - Con el protocolo 2PL estricto el plan se ejecuta de la siguiente manera:

T_1	T_2
$X(A)$	
$R(A)$, A=1000	
$W(A)$, A=900	
$X(B)$	
$R(B)$= 500	
$W(B)$= 600	
$commit_{T_1}$	
	$X(A)$
	$R(A)$, A=900
	$W(A)$, A=990
	$X(B)$
	$R(B)$, B=600
	$W(B)$, B=660
	$commit_{T_2}$

Administración de Transacciones

- Ejemplo: Protocolo 2PL Estricto
  - El protocolo 2PL estricto también permite la ejecución de transacciones intercaladas:

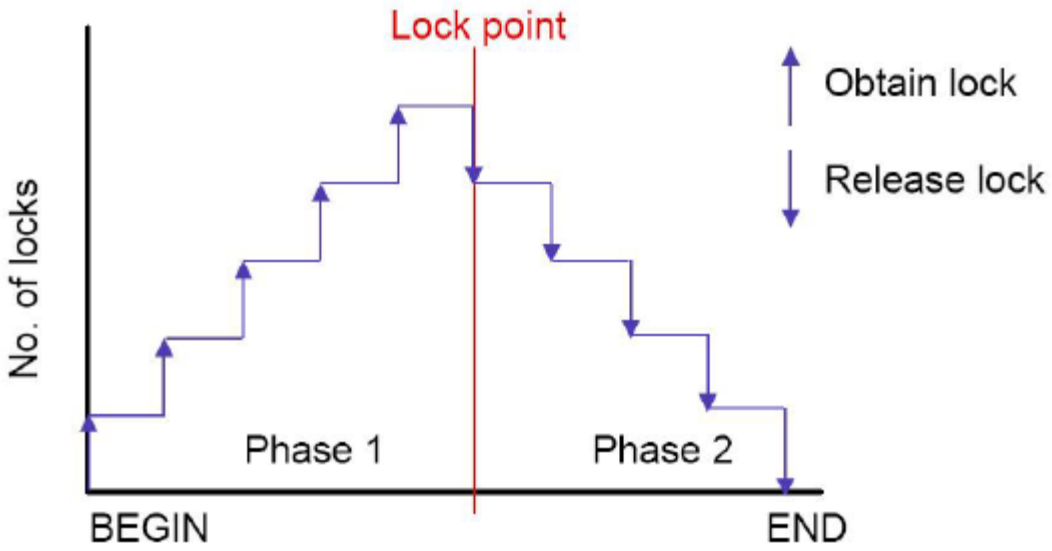


Administración de Transacciones

- Protocolo 2PL
  - Existe una variante del protocolo 2PL estricto llamado 2PL que relaja la segunda regla del protocolo 2PL estricto
  - En 2PL una transacción puede entregar sus candados antes del final de la transacción (antes de commit o abort)
  - En 2PL la segunda regla es: Una transacción no puede pedir candados adicionales una vez que empieza a devolver sus candados
  - El protocolo 2PL asegura que los grafos de dependencias de los planes sean acíclicos y por lo tanto solo permite planes conflicto serializables

Administración de Transacciones

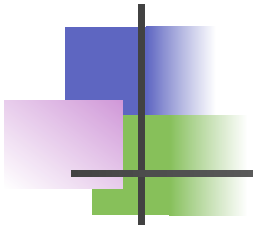
- Protocolo 2PL



Administración de Transacciones

- Ejemplo Protocolo 2PL
  - Un plan aceptado por 2PL pero no por 2PL estricto:

T_1	T_2
$X(A)$	
$R(A)$	
$W(A)$	
$X(B)$	
libera $X(A)$	
	$X(A)$
	$R(A)$
	$W(A)$
	$X(C)$
	$W(C)$
	libera $X(A)$
	libera $X(C)$
	$commit_{T_2}$
$W(B)$	
libera $X(B)$	
$commit_{T_1}$	

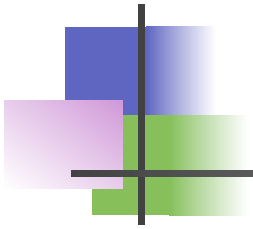


Administración de Transacciones

- Soporte de Transacciones en SQL
  - En SQL una transacción es inicializada cada vez que un usuario ejecuta una sentencia SELECT, UPDATE, o CREATE TABLE
  - SQL también permite bloquear objetos con diferentes niveles de granularidad, e.g. tuplas en vez de tablas
  - Supongamos la siguiente transacción T_1(en SQL):

```
SELECT MIN(edad)
FROM Navegantes
WHERE categoria = 8
```

  - ¿Qué objetos deberían ser bloqueados?



Administración de Transacciones

- Soporte de Transacciones en SQL
  - Supongamos que SQL permite bloquear todas las tuplas que satisfacen la condición categoria= 8. Esto no previene que otra transacción T2 inserte una nueva tupla que satisfaga la condición
  - Entonces, si T_1 selecciona nuevamente las tuplas con categoría igual a 8 el resultado será diferente
  - Tal problema se denomina problema fantasma, en donde una transacción ejecuta una consulta dos veces y obtiene diferentes conjuntos de tuplas, a pesar de no haber modificado las tuplas
  - Para prevenir este problema el SGBD debe bloquear todas las posibles tuplas que podrían estar involucradas en la transacción T_1, entonces, la solución en este caso es bloquear toda la tabla, pagando el costo de concurrencia

Administración de Transacciones

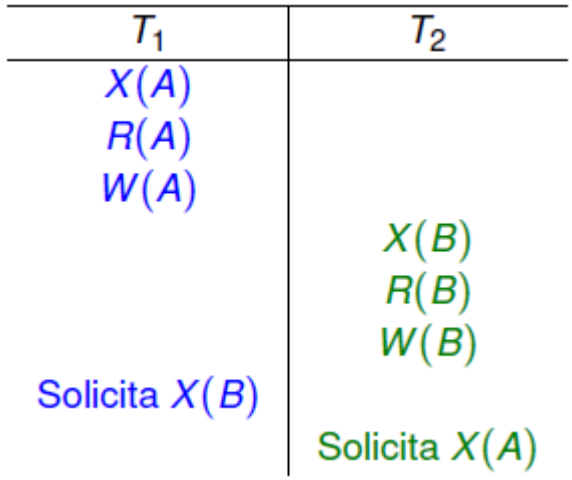
- Soporte de Transacciones en SQL
  - SQL permite elegir algunos niveles de aislamiento:
    - READ UNCOMMITTED
    - READ COMMITTED
    - REPEATABLE READ
    - SERIALIZABLE
  - Con los siguientes efectos sobre las transacciones:

Nivel	Lectura sucia	Lectura no repetida	P. Fantasma
READ UNCOMMITTED	es posible	es posible	es posible
READ COMMITTED	NO	es posible	es posible
REPEATABLE READ	NO	NO	es posible
SERIALIZABLE	NO	NO	NO

  - http://www.postgresql.org/docs/9.1/static/transaction-iso.html

Administración de Transacciones

- Interbloqueos
  - Supongamos la siguiente situación:



  - Estos ciclos entre transacciones se denominan interbloqueos
  - El SGBD debe prevenir o detectar (y resolver) estas situaciones

Administración de Transacciones

- Detección de Interbloqueos
  - El administrador de candados en un SGBD es el encargado de administrar los permisos sobre objetos de la BD
  - El administrador mantiene un grafo de esperas G(T) para detectar interbloqueos que se construye de la siguiente manera:
    - Cada transacción activa es un nodo en G(T)
    - Existe un arco desde T_i a T_j sí y solo sí T_i está esperando que T_j libere un candado
  - El administrador agrega arcos al grafo cada vez que una transacción pide un candado y elimina los arcos cuando las transacciones obtienen los candados

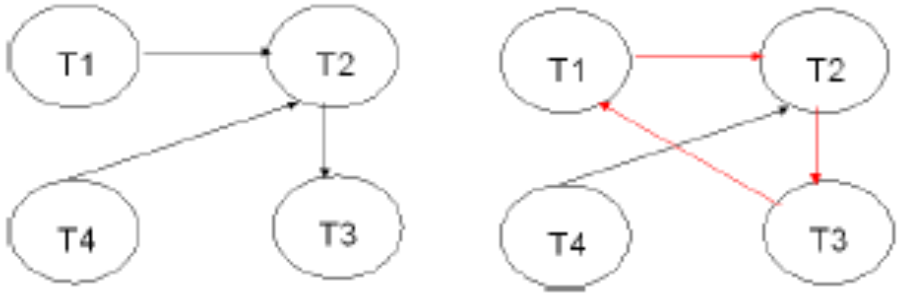
Administración de Transacciones

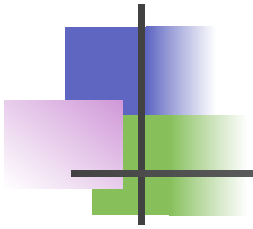
- Ejemplo Detección de Interbloqueos

  - Supongamos la siguiente situación:

T_1	T_2	T_3	T_4
$S(A)$			
$R(A)$			
	$X(B)$		
	$W(B)$		
$S(B)$			
		$S(C)$	
		$R(C)$	
	$X(C)$		
			$X(B)$
		$X(A)$	

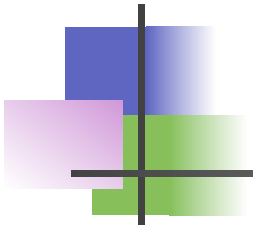
  - El Grafo G(T) antes y después del interbloqueo:





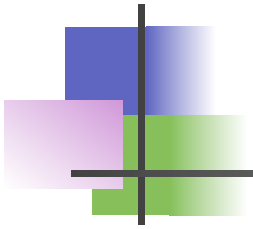
Administración de Transacciones

- Solución de Interbloqueos
  - El problema se resuelve abortando una de las transacciones involucradas en el ciclo y liberando todos sus candados
  - La elección de la transacción a abortar puede realizarse en base a varios criterios:
    1. La transacción que posee menos candados
    2. La transacción que ha avanzado menos en su ejecución
    3. La transacción que dista mucho de su finalización
    4. etc.



Administración de Transacciones

- Prevención de Interbloqueos
  - Se le otorga a cada transacción un nivel de prioridad
  - Si una transacción T_i requiere un candado para el objeto A y una transacción T_j mantiene un candado conflictivo para A el administrador de candados puede seguir cualquiera de las siguientes políticas:
    1. Si T_i tiene menor prioridad que T_j , entonces T_i espera a que T_j libere el candado. En caso contrario, T_i aborta
    2. Si T_i tiene mayor prioridad que T_j , entonces T_j se aborta. En caso contrario, T_i espera

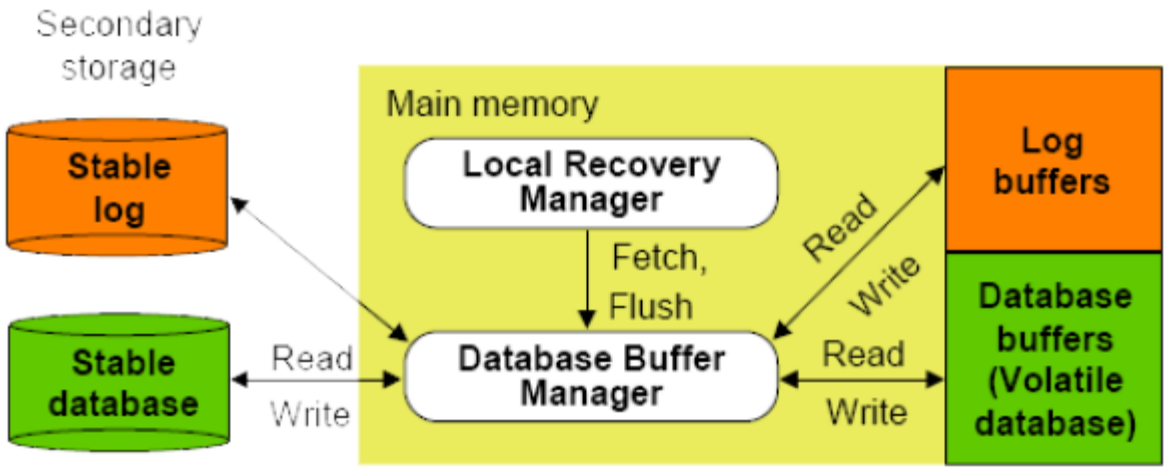


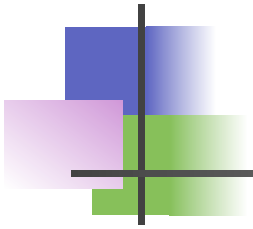
Administración de Transacciones

- Introducción
- Transacciones
- Control de Concurrencia
- Recuperación de BD

Recuperación de BD

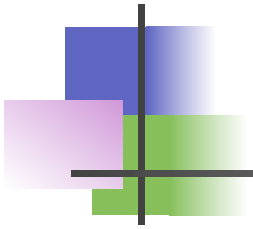
- El administrador de recuperación de un SGBD es el encargado de asegurar que las transacciones sean atómicas y durables





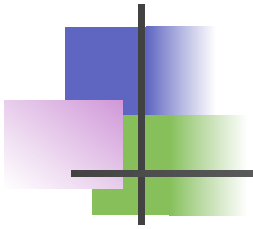
Recuperación de BD

- Tipos de fallas:
  - Caída del sistema
  - Error de la transacción o del sistema
  - Condiciones de excepción detectados por la transacción
  - Control de la concurrencia
  - Falla del disco
  - Problemas físicos y catástrofe (alimentación de energía por ejemplo)



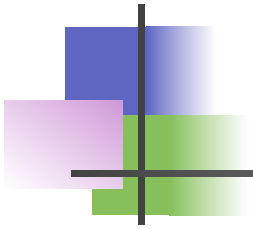
Recuperación de BD

- LOG
  - El LOG es un historial de las acciones ejecutadas por el SGBD
  - El LOG es un archivo que se almacena en disco y generalmente se mantienen varias copias del LOG en diferentes discos
  - Una parte del LOG, la que contiene las operaciones recientes, se almacenada en memoria principal
  - Cada registro en el LOG tiene un identificador, el LSN (Log Sequence Number)



Recuperación de BD

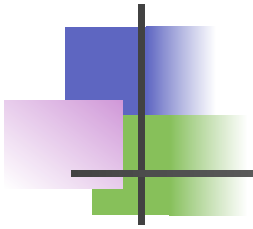
- LOG
  - Las siguientes operaciones se registran en el LOG:
    - Updates: Actualizaciones a páginas
    - Commit: una transacción compromete sus cambios una vez que commit es registrado en el LOG y éste ha sido almacenado en disco
    - Abort
    - End: finalización real de la transacción, se finalizan operaciones extras como borrar archivos, etc.
    - Recuperación: se registran todas las actualizaciones que son deshechas debido a acciones abort de transacciones o fallas del sistema



Recuperación de BD

- Algoritmo de recuperación ARIES

  - El administrador de recuperación es invocado después de una caída del sistema. ARIES es un algoritmo simple que trabaja con una amplia gama de mecanismos de control de concurrencia y está implementado en varios SGBD



Recuperación de BD

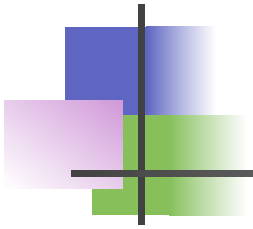
- Algoritmo de recuperación ARIES
  - El proceso de recuperación de ARIES tiene tres fases:
    1. Analisis: identificación de "páginas sucias" (páginas con datos en el buffer que no han sido escritas a disco) y la identificación de las transacciones activas
    2. Rehacer: repetir todas las acciones partiendo desde un punto apropiado del LOG para dejar la BD en el mismo estado que estaba antes de la caída
    3. Deshacer: deshacer todas las acciones de transacciones que no se comprometieron. De esta manera, la BD refleja solo las acciones de transacciones comprometidas

Recuperación de BD

- Algoritmo de recuperación ARIES
  - Considere el siguiente historial

LSN		LOG
10	–	update: T_1 escribe $P5$
20	–	update: T_2 escribe $P3$
30	–	T_2 commit
40	–	T_2 end
50	–	update: T_3 escribe $P1$
60	–	update: T_3 escribe $P3$
	X	CRASH, RESTART

  - Análisis: transacciones activas: T_1 y T_3, las cuales deben ser deshechas. Páginas con información "sucia": P1;P3;P5
  - Rehacer: todos los "updates" son aplicados nuevamente en el orden mostrado en el historial (incluyendo updates de T_1 y T_3)
  - Deshacer: T_1 y T_3 son deshechas en orden reverso, i.e. primero las escrituras de T_3, y luego la escritura de T_1



Recuperación de BD

- Principios de ARIES
  - Primero escribir el LOG: primero se registra en el LOG cualquier cambio a los objetos de la BD. El LOG se almacena en disco antes de modificar la BD en el disco
  - Repetir todas las acciones en el historial: cuando comienza la recuperación, ARIES repite todas las acciones ocurridas antes de la caída. De esta forma, el sistema vuelve al mismo estado en que se encontraba antes de la falla. Luego se deshacen todas las acciones de las transacciones activas
  - Registrar en el LOG los cambios durante la recuperación: todos los cambios hechos durante la recuperación del sistema son registrados en el LOG (para evitar repetir acciones en caso de otra caída)

Recuperación de BD

- Caídas durante el proceso de recuperación
  - ¿Qué pasa si hay una caída durante el proceso de recuperación del sistema?

LSN		LOG
10	–	update: T_1 escribe $P5$
20	–	update: T_2 escribe $P3$
30	–	T_1 abort
40,45	–	Deshacer T_1 LSN 10, T_1 end
50	–	update: T_3 escribe P_1
60	–	update: T_2 escribe P_5
	X	CRASH, RESTART
70	–	Deshacer T_2 LSN 60
80,85	–	Deshacer T_3 LSN 50, T_3 end
	X	CRASH, RESTART
90,95	–	Deshacer T_2 LSN 20, T_2 end

Recuperación de BD

- Ejercicios:
  - Para los siguientes historiales explique las acciones realizadas en la etapa de análisis, rehacer y deshacer y complete el historial con las acciones efectuadas en la etapa de deshacer

LSN		LOG
20	–	update: T_1 escribe $P5$
30	–	update: T_2 escribe $P3$
40	–	T_2 commit
50	–	T_2 end
60	–	update: T_3 escribe $P3$
70	–	T_1 abort
	X	CRASH, RESTART

LSN		LOG
20	–	update: T_1 escribe $P1$
30	–	update: T_2 escribe $P2$
40	–	update: T_3 escribe $P3$
50	–	T_2 commit
60	–	update: T_3 escribe $P2$
70	–	T_2 end
80	–	update: T_1 escribe $P5$
90	–	T_3 abort
	X	CRASH, RESTART