Administración de Transacciones

Base de Datos

Mónica Caniupán
mcaniupan@ubiobio.cl

Universidad del Bío-Bío

2020

Transacciones

- Las transacciones son el fundamento de la ejecución concurrente y la recuperación de un fallo del sistema en un SGBD
- Una transacción se define como cualquier ejecución de un programa de usuario en un SGBD
- Un SGBD entrelaza las acciones de varias transacciones y debe garantizar que el resultado de una ejecución concurrente de transacciones sea equivalente (en su efecto sobre la BD) a alguna ejecución secuencial

Propiedades de las Transacciones

- Desde el punto de vista de un SGBD, una transacción es una serie de operaciones de escritura y lectura a la BD
- Un SGBD debe garantizar cuatro propiedades:
  1. Atómicas: la ejecución de cada transacción es atómica, es decir, o se realizan todas las acciones o no se realiza ninguna
  2. Consistencia: las transacciones deben preservar la consistencia de la BD
  3. Aislamiento: las transacciones deben estar aisladas (protegidas) de los efectos de otras transacciones
  4. Durabilidad: si la transacción finaliza exitosamente, sus efectos deben persistir incluso si se produce una caída del sistema

Acciones de una Transacción

- Las acciones que puede ejecutar una transacción son: lectura y escritura de objetos de la BD
  - $R_t(O)$ denota lectura del objeto O por la transacción t
  - $W_t(O)$ denota escritura de O por t
  - Cada transacción debe indicar como última acción, una de las siguientes:
    - commit (comprometer): indica que la transacción termina satisfactoriamente
    - abort (abortar): indica que la transacción termina pero se deshacen los cambios realizados a la BD

Planes

- Un plan de ejecución (planificación) es una lista de acciones (lectura, escritura, comprometer o abortar) de un conjunto de transacciones
- El orden en el cual dos acciones de una transacción T aparecen en un plan debe ser el mismo orden en el que aparecen en T

Ejemplo: Plan

- Consideremos los siguientes planes:

Plan serial: $T_1 - T_2$	
T_1	T_2
$R(A)$	
$W(A)$	
$R(C)$	
$W(C)$	
$commit_{T_1}$	
	$R(B)$
	$W(B)$
	$commit_{T_2}$

Plan no serial	
T_1	T_2
$R(A)$	
$W(A)$	
	$R(B)$
	$W(B)$
	$commit_{T_2}$
$R(C)$	
$W(C)$	
$commit_{T_1}$	

- Si las acciones de diferentes transacciones no son intercaladas, i.e. las transacciones son ejecutadas de inicio a fin, una por una, el plan se denomina plan serial
- Para el ejemplo existen dos posibles planes seriales: $T_1 - T_2$ y $T_2 - T_1$

Ejecución Concurrente de Transacciones

- Los SGBD entrelazan las acciones de distintas transacciones para mejorar el rendimiento
  - Si una transacción necesita leer desde disco, el SGBD podría empezar a procesar otra transacción
- Surge entonces el concepto de planes serializables
- Un plan serializable sobre un conjunto S de transacciones que comprometen los cambios, es un plan cuyo efecto sobre cualquier instancia consistente de la BD se garantiza idéntico al de alguna planificación secuencial sobre S

Ejemplo: Planes Serializables

- Los siguientes planes son serializables, el primero produce el mismo efecto que ejecutar $T_1 - T_2$, el segundo es equivalente a $T_2 - T_1$

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

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

- Las ejecuciones secuenciales pueden producir distintos resultados, pero todos se suponen aceptables; el SGBD no garantiza cuál de ellos será el resultado de una ejecución entrelazada

Problemas al Intercalar Transacciones

- Dos acciones sobre el mismo objeto de la BD presentan conflicto si por lo menos una de las acciones es una escritura
- Existen tres tipos de conflictos entre dos transacciones T_1 y T_2:
  1. Conflicto de escritura-lectura o lectura de datos no comprometidos
  2. Conflicto de lectura-escritura o lecturas no repetidas
  3. Conflicto de escritura-escritura o sobre-escritura de datos no comprometidos

(1) Conflicto de escritura-lectura

- El principal problema de escritura-lectura es la lectura de datos no comprometidos, i.e. una transacción T_2 lee un objeto que fue modificado por una transacción T_1, pero que aún no ha sido comprometida
- En tal caso, la lectura es conocida como lectura sucia

Ejemplo: Lectura Sucia

- Consideremos las transacciones T_1 y T_2:
  - T_1 transfiere 100 desde la cuenta A a la B
  - T_2 incrementa A y B en un 10 %
- Supongamos los valores iniciales $A = 1000$ y $B = 500$

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	
$commit_{T_1}$	

- Este plan produce los valores $A = 990$ y $B = 650$
- $T_1 - T_2$ produce los valores $A = 990$ y $B = 660$
- $T_2 - T_1$ produce $A = 1000$ y $B = 650$
- El problema es que T_2 lee el valor de A modificado por T_1 pero que no ha sido comprometido

(2) Conflicto de lectura-escritura

- El principal problema de lectura-escritura es la lectura no repetida de datos, i.e. una transacción T_2 modifica el valor de un objeto A que es leído por una transacción T_1 y T_1 sigue en progreso
- Si T_1 vuelve a leer A, el valor será diferente
- Consideremos las transacciones T_1 y T_2, y sea A el número de copias disponibles de un libro
  - T_1 lee $A = 1$
  - T_2 lee $A = 1$, resta 1 y compromete (A=0)
  - T_1 trata de restar 1 a A y recibe un error, A ya no tiene el valor 1!

(3) Conflicto de escritura-escritura

- Se da cuando una transacción T_2 sobre-escribe el valor de un objeto A que ya ha sido modificado por una transacción T_1, mientras T_1 todavía está en progreso
- Este problema se conoce como sobre-escritura de datos no comprometidos
- Supongamos que los salario de Pedro y Juan deben ser iguales
  - T_1 asigna salarios iguales a 2000
  - T_2 asigna salarios iguales a 1000
- El plan $T_1 - T_2$ asigna salarios iguales a 1000, $T_2 - T_1$ asigna salarios iguales a 2000

Conflicto de escritura-escritura

- Supongamos la siguiente ejecución intercalada:

T_1	T_2
	$W(Pedro)$, Salario=1000
$W(Juan)$, Salario=2000	
	$W(Juan)$, Salario=1000
	$commit_{T_2}$
$W(Pedro)$, Salario=2000	
$commit_{T_1}$	

- El plan no produce salarios iguales para Juan y Pedro
- Por lo tanto este plan no es serializable

Planes con Abort

- Cuando una transacción aborta, todas las acciones son deshechas y la BD vuelve al estado inicial
- Un plan serializable sobre un conjunto de transacciones S es un plan cuyo efecto sobre cualquier BD consistente es idéntico a alguna ejecución serial sobre el conjunto de transacciones comprometidas en S
- Esta definición de plan serializable asume que las transacciones que abortan deben ser completamente deshechas, lo cual puede ser imposible en algunas situaciones

Ejemplo: Planes con Abort

- Consideremos las transacciones T_1 y T_2:
  - T_1 transfiere 100 desde la cuenta A a la B
  - T_2 incrementa A y B en un 10 %
- Supongamos los valores iniciales $A = 1000$ y $B = 500$

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
	$commit_{T_2}$
$abort_{T_1}$	

- T_2 ha leído un valor de A que no debería haber estado nunca ahí
- Si T_2 no comprometiera antes que T_1 aborta, podríamos tratar la situación propagando el aborto de T_1 en cascada y abortar T_2
- Pero T_2 ya comprometió y sus acciones no pueden ser deshechas! Este plan no es recuperable

Planes Recuperables

- En un plan recuperable las transacciones se comprometen sólo después de que se comprometen todas las transacciones de las cuales han leído cambios
- Si las transacciones leen solamente los cambios de transacciones comprometidas, el plan no sólo es recuperable, sino que al abortar una transacción no se necesita que otras transacciones aborten en cascada (planes que evitan abortos en cascada)

Planes Conflicto Equivalentes

- Dos planes son conflicto equivalentes si:
  - Involucran el mismo conjunto de acciones de las mismas transacciones, y
  - Ordenan cada par de acciones conflictivas de dos transacciones que comprometen de la misma manera
- Dos acciones están en conflicto si operan sobre el mismo objeto y al menos una de ellas es escritura
- Si dos planes son conflicto equivalentes, tienen el mismo efecto sobre la BD.
- Para saber si dos planes cumplen con esta propiedad, podemos cambiar el orden de acciones que no son conflictivas sin alterar el efecto del plan sobre la BD
- Un plan es conflicto serializable si es conflicto equivalente con algún plan serial
- Todo plan conflicto equivalente es serializable

Ejemplo: Planes Conflicto Equivalentes

- Consideremos el siguiente plan S:
  $R_1(x), R_2(y), W_3(x), R_2(x), R_1(y)$
- Las operaciones pueden ser re-ordenadas de la siguiente manera:
  - Las lecturas de T_1 pueden ser ejecutadas en el siguiente orden:
    $R_1(x), R_1(y), R_2(y), W_3(x), R_2(x)$
  - La lectura de T_2 se puede reordenar de la siguiente forma:
    $R_1(x), R_1(y), W_3(x), R_2(y), R_2(x)$
  - Por lo tanto, hemos demostrado que el plan es conflicto equivalente a la ejecución serial T_1-T_3-T_2, y como consecuencia el plan es serializable
- Es decir, el plan $S : R_1(x), R_2(y), W_3(x), R_2(x), R_1(y)$ ejecutado de esta forma, produce el mismo efecto que la ejecución serial T_1-T_3-T_2

Ejemplo: Planes Conflicto Equivalentes

- Consideremos el siguiente plan:
  $R_2(y), R_1(x), R_3(y), R_2(x), W_2(y), W_1(x), R_3(x)$
- Podemos reordenar las operaciones de la siguiente manera:
  - La lectura de T_1 puede ser localizada junto con la escritura de T_1
    $R_2(y), R_3(y), R_2(x), W_2(y), R_1(x), W_1(x), R_3(x)$
  - Luego, las operaciones de T_2 se reordenan de la siguiente forma:
    $R_3(y), R_2(y), R_2(x), W_2(y), R_1(x), W_1(x), R_3(x)$
  - Sin embargo, no podemos mover $R_3(y)$ ni $R_3(x)$ sin causar conflictos, por lo tanto este plan no es conflicto equivalente a ninguna ejecución serial de las transacciones

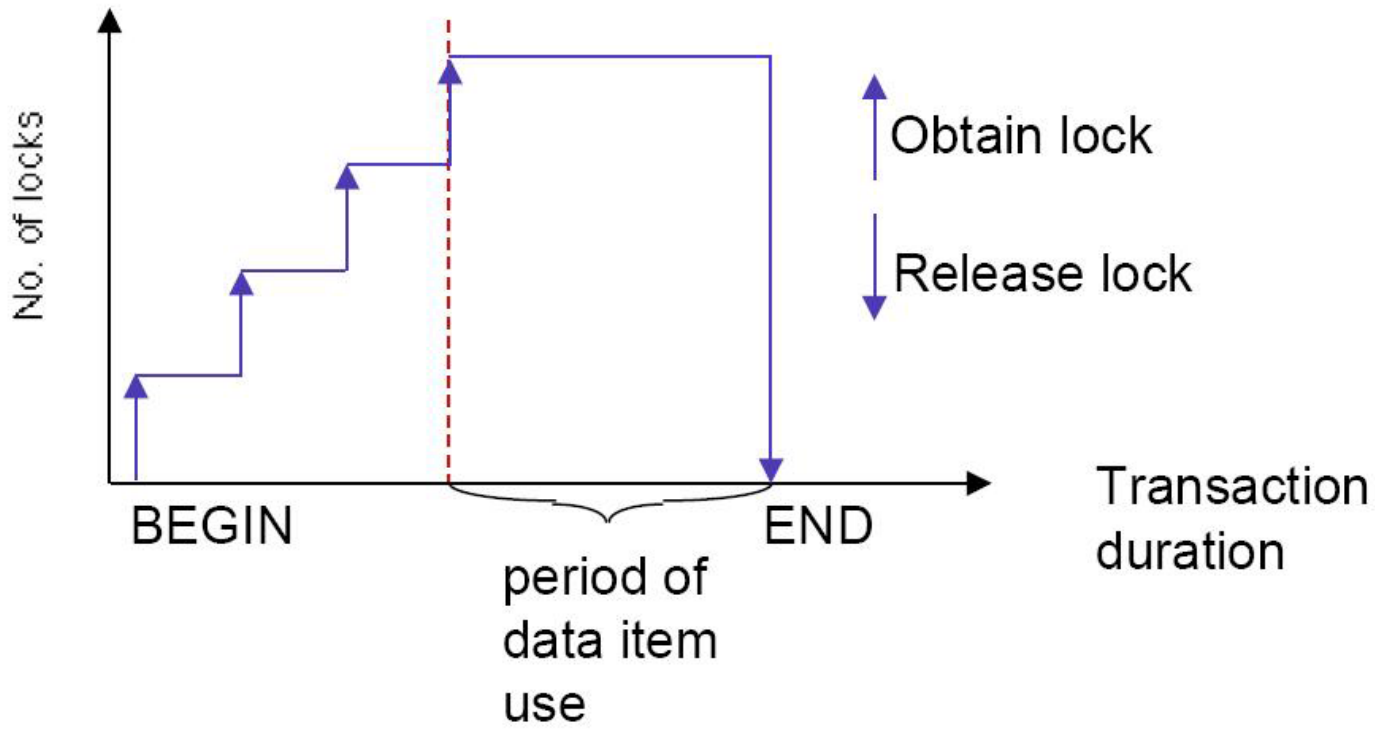
Control de Concurrencia Basado en Candados (Bloqueos)

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

Bloqueo Exclusivo de Dos Fases (2PL Estricto)

- El protocolo 2PL exclusivo (estricto) tiene dos reglas:
  1. Si una transacción T quiere leer (respectivamente, modificar) un objeto, solicita un candado (bloqueo) compartido (respectivamente, exclusivo) sobre el objeto
  2. Todos los candados concedidos a una transacción se liberan cuando la transacción se completa
- Obviamente, si una transacción tiene un candado exclusivo sobre un objeto, también puede leer el objeto. Por lo tanto, si una transacción va a escribir eventualmente sobre un objeto, se recomienda que pida de inmediato el candado exclusivo
- Una transacción que requiere un candado espera hasta que el SGBD pueda otorgar el candado

Bloqueo Exclusivo de Dos Fases (2PL Estricto)



Bloqueo Exclusivo de Dos Fases (2PL Estricto)

- 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:

Tipo de Candado	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)

Ejemplo: Protocolo 2PL Estricto

- Consideremos las transacciones T_1 y T_2:
  - T_1 transfiere 100 desde la cuenta A a la B
  - T_2 incrementa A y B en un 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	
$commit_{T_1}$	

Ejemplo: Protocolo 2PL Estricto

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

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	
$commit_{T_1}$	

T_1	T_2
$X(A)$	
$R(A)$, A=1000	
$W(A)$, A=900	$X(A)$ ...(espera)
$X(B)$	
$R(B)$= 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
	$commit_{T_2}$

Ejemplo: Protocolo 2PL Estricto

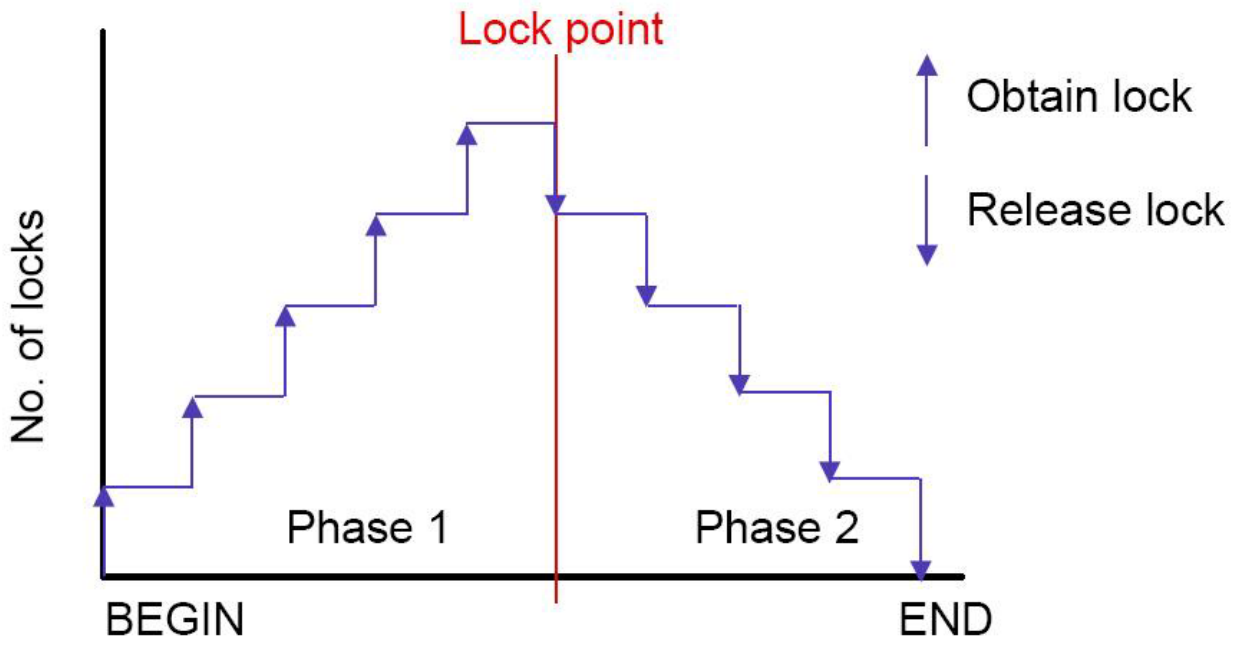
- El protocolo 2PL estricto también permite la ejecución de transacciones intercaladas:

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

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

Protocolo 2PL



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 $X(B)$	
$commit_{T_1}$	

Interbloqueos

- Supongamos la siguiente situación:

T_1	T_2
$X(A)$	
$R(A)$	
$W(A)$	
	$X(B)$
	$R(B)$
	$W(B)$
Solicita $X(B)$	
	Solicita $X(A)$

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

Interbloqueos

- Una forma usual de resolver los interbloqueos es usando un mecanismo de tiempos, i.e., si una transacción ha esperado un candado por mucho tiempo, se asume que ha ocurrido un interbloqueo y la transacción se aborta
- En la práctica menos del 1 % de las transacciones se ve involucrada en un interbloqueo
- Los interbloqueos pueden prevenirse:
  - Bloqueando partes de los objetos, e.g. tuplas en vez de tablas completas
  - Reduciendo el tiempo en que una transacción puede mantener un candado
  - Reduciendo la cantidad de objetos que son leídos y modificados

Detección de Interbloqueos

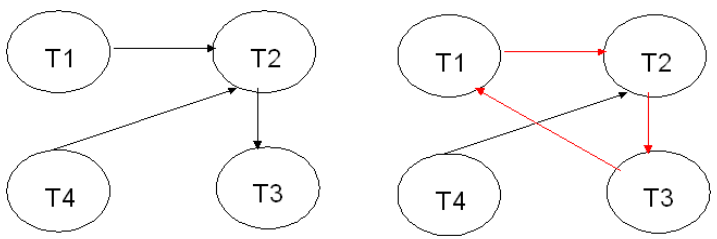
- El administrador de candados en un SGBD es el encargado de administrar los permisos sobre objetos de la BD
- El administrador mantiene un grafo de esperas $\mathcal{G}(T)$ para detectar interbloqueos que se construye de la siguiente manera:
  - Cada transacción activa es un nodo en $\mathcal{G}(T)$
  - 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

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 $\mathcal{G}(T)$ antes y después del interbloqueo:



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.

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 mayor 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