

A decorative graphic on the left side of the slide, consisting of a network of white lines and circles on a blue background, resembling a circuit board or a tree structure.

CONTROL DE CONCURRENCIA

ÍNDICE

- **Introducción**

- Lock binario, shared lock
- Two phase locking
- Control de concurrencia por timestamp
- Control de concurrencia por multiversion
- Niveles Aislamiento en SQL

INTRODUCCIÓN

- Existen 3 problemas básicos de control de concurrencia
 - Modificación perdida
 - Falsa Sumarizacion
 - Falsa Modificación (o dirty read)
- Cuando el scheduler recibe una operación puede tomar 3 decisiones
 - Rechazarla
 - Procesarla
 - Demorarla
- Si el scheduler usualmente favorece el procesamiento de la operación se lo llama “agresivo”, sino se lo llama “conservador”

ÍNDICE

- Introducción
- **Lock binario, shared lock**
- Two phase locking
- Control de concurrencia por timestamp
- Control de concurrencia por multiversion
- Niveles Aislamiento en SQL

LOCK BINARIO – SHARED LOCK

- Estos dos mecanismos utilizan el concepto de “lock”, es decir asocian una variable a cada “data ítem” para indicar si el mismo esta disponible o no
- El lockeo binario es el modelo más sencillo. Un data ítem simplemente puede estar lockeado o no lockeado
- La notación es
 - $l_i(x)$: T_i lockea el data ítem x
 - $u_i(x)$: T_i libera el lock sobre el data ítem x
- Cuando una transacción lee o escribe un ítem solicita un lock sobre el mismo
- Como se resuelve el problema de la falsa modificación con esta aproximación?
- Que problema tiene esta aproximación?

LOCK BINARIO – SHARED LOCK (CONT)

- ▶ Para resolver la limitación del lockeo binario el shared lock o lockeo ternario permite que el lock sea exclusivo o compartido (shared)
- ▶ La notación es
 - ▶ $rl_i(x)$: Ti lockea el data ítem x en formato compartido
 - ▶ $wl_i(x)$: Ti lockea el data ítem x en formato exclusivo
- ▶ Son posibles las siguientes situaciones?
 1. $rl1(x), rl2(x)$
 2. $wl1(x), wl2(x)$
- ▶ Existe la posibilidad de que una transacción efectúe un “upgrade” de un lock para pasar de un lock de lectura a uno de escritura
- ▶ La notación es $ul_i(x)$

LOCK BINARIO – SHARED LOCK (CONT.)

		Lock Solicitado	
		Shared	Exclusive
Lock existente	Shared	OK	No ok
	Exclusive	No ok	No ok

ÍNDICE

- Introducción
 - Lock binario, shared lock
 - **Two phase locking**
 - Control de concurrencia por timestamp
 - Control de concurrencia por multiversion
- Niveles Aislamiento en SQL

TWO PHASE LOCKING -INTRO

- Es un mecanismo muy simple, en el cual cada data ítem tiene un lock que puede estar o no activo
- La notación es la misma que la utilizada para el shared lock
- En que situación hay un conflicto entre los locks?

TWO PHASE LOCKING (2 PL) - BASICO

- Dos locks $ol_i(x)$ y $pl_i(x)$ entran en conflicto si $i \neq j$, y uno de los dos se refiere a un write
- Un 2 PL básico respeta las siguientes reglas
 - 1. Cuando recibe una operación $pi[x]$ del Transaction Manager (TM) verifica si $pi[x]$ entra en conflicto con alguna operación $qlj[x]$ que ya haya ocurrido. Si esto sucede demora $pi[x]$, obligando a T_i a esperar hasta que el lock se libere. Si $pi[x]$ no entra en conflicto con ninguna operación que ya haya ocurrido envía $pi[x]$ al Data Manager (DM).
 - 2. Una vez que el scheduler estableció un lock para T_i , por ejemplo $pli[x]$, no lo va a liberar hasta que el DM informe que proceso la operación correspondiente al lock, $pi[x]$.
 - 3. Una vez que el scheduler libero un lock de una transacción no va a adquirir ningún nuevo lock para esa transacción

TWO PHASE LOCKING - EJEMPLO

- La regla 3 es la que le da el nombre al mecanismo, dado que establece una fase de “crecimiento” (al tomar los locks) y una de “decrecimiento” (al liberar los locks)
- Tomemos el siguiente ejemplo
 - $H = r1(x), r1(x), u1(x), w12(y), w2(y), u2(y), r1(y), r1(y), u1(y), c2, c1$
 - Es serializable?
 - Que tipo de Schedule es ?
 - Cumple con las reglas del 2PL?
- Este tipo de mecanismos no evitan el deadlock. Pueden pensar un ejemplo?

TWO PHASE LOCKING - DEADLOCK

- La existencia de deadlocks implica que se necesita una estrategia para tratarlos. Una posibilidad es establecer un timeout, pero cuanto es un buen timeout?
- Otra posibilidad es construir un grafo para representar que operaciones “esperan” de otras. Este grafo se llama Wait For Graph (WFG). Se genera un nodo por cada transacción y un arco entre 2 nodos cuando una transacción espera a otra. Un deadlock se produce cuando hay un ciclo en ese grafo
- El Scheduler va a agregar un arco de T_i a T_j cada vez hay un lock de T_i esperando a que se libere un lock de T_j . El arco se libera cuando se libera el ultimo lock de T_j que espera T_i
- Cual es el WFG del ejemplo que propusieron?
- En que momento se efectúa el control de ciclos del grafo?
- Como se elije a la victima?

TWO PHASE LOCKING - CORRECTITUD

- Para que el algoritmo sea correcto es necesario probar que las historias que produce son serializables
- De la regla 1 de la definición de 2PL se deduce que
 - Proposición 1: Sea H es una historia producida por un 2PL. Si $oi(x)$ esta en $C(H)$, entonces $oli(x)$ y $ui(x)$ están en $C(H)$ y $oli(x) < oi(x) < ui(x)$
 - Proposición 2: Sea H una historia producida por un 2PL. Si $pi(x)$ y $qj(x)$ ($i < > j$) son dos operaciones en conflicto en $C(H)$, entonces $ui(x) < qli(x)$ o $uj(x) < qli(x)$
- De la regla 3 de la definición se deduce que
 - Proposición 3: Sea H es una historia producida por un 2PL. Si $pi(x)$ y $qi(y)$ están en $C(H)$, entonces $pli(x) < ui(y)$

TWO PHASE LOCKING – CORRECTITUD 2

- Usando esto vamos a demostrar que el grafo que produce el 2LPL es acíclico
- 1) Si $T_i \rightarrow T_j$ esta en $SG(H)$, entonces una de las operaciones de T_i en un data ítem, por ejemplo x , se ejecuto antes y entro en conflicto con una de las operaciones de T_j . Entonces T_i debe liberar el lock de x antes de que T_j pueda tomarlo.
- 2) Supongamos $T_i \rightarrow T_j \rightarrow T_k$ es un camino en $SG(H)$, del primer paso se deduce que T_i libero algún lock antes de que T_j lo tomara y lo mismo vale entre T_j y T_k . Por lo tanto por transitividad T_i libero un lock antes de que T_k tomara uno. Por inducción esto puede extenderse arbitrariamente
- 3) Supongamos que $SG(H)$ tiene un ciclo, digamos que $T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_n \rightarrow T_1$. Por el punto anterior esto implicaría que T_1 libero un lock antes de que T_1 tomara uno, lo que contradice la tercera regla del protocolo.
- Por lo tanto el grafo no puede tener ciclos y H es serializable

ÍNDICE

- Introducción
 - Lock binario, shared lock
 - Two phase locking
 - **Control de concurrencia por timestamp**
 - Control de concurrencia por multiversion
- Niveles Aislamiento en SQL

CONTROL DE CONCURRENCIA POR TIMESTAMP

- Este es un mecanismo de control de concurrencia que no lockea
- El TM le asigna a cada operación un timestamp. Todas las operaciones que componen esa transacción llevan el mismo timestamp
- Regla: Si $p_i(x)$ y $q_j(x)$ son dos operaciones en conflicto, entonces $p_i(x) < q_j(x)$ si y solo si $TS(T_i) < TS(T_j)$
- Una forma de implementar el control por Timestamp es que el scheduler envía la operación al DM a menos que esta llegue “muy tarde”.
- Una operación $q_j(x)$ “llega tarde” cuando el scheduler ya procesó $o_i(x)$, con $TS(T_i) < TS(T_j)$ y ambas operaciones están en conflicto. En este caso el scheduler aborta T_i y la relanza, con lo cual su timestamp será posterior.

TIMESTAMP - CONT

- Para poder verificar si una operación llega tarde , el scheduler mantiene para cada data ítem su max-r-scheduler y su max-w-scheduler.
- Al recibir una operación compara su timestamp con los “maxs” correspondientes para indicar si la misma puede continuar o no.
- Si su timestamp es menor que algunos de los que corresponden a operaciones en conflicto la operación es cancelada
- Que operaciones controlan con que max? Por que?

	Max-r-scheduler	Max-w-scheduler
read	NO	si
write	SI	SI

- Si la operación supera el control, el scheduler actualiza el TS correspondiente y programa la operación

TIMESTAMP – THOMAS WRITE RULE

- Supongamos que tenemos dos transacciones $T1$ y $T2$ con $TS(T1) < TS(T2)$, y supongamos que se produce la siguiente situación
 - $w2(x) \ w1(x)$
- Este plan es valido para el control por timestamp?
- Existe alguna solución?
- Una posibilidad es “ignorar” el $w1(x)$. Por que es valida esta posibilidad?
- La “Thomas Write Rule” establece :
 - Si $max-w-scheduler(x) = TS(Tj)$ y se recibe la operación $wi(x)$ con $TS(Ti) < TS(Tj)$ no procesar $wi(x)$, darla simplemente por “cumplida”

THOMAS WRITE RULE (CONT)

- Supongamos ahora que se produce la siguiente situación
 - $R_2(x) \ w_2(x) \ w_1(x)$
- En este plan puede aplicarse la TWR?
- Si, puede aplicarse, sin embargo el resultado no es correcto dado que T2 lee antes de escribir y lo que escribe seguramente este influido por el valor de x. En este caso al no haber ejecutado $w_1(x)$ previamente a esa lectura el resultado puede ser diferente.
- En realidad el control por timestamp no permite que esta situación se produzca, dado que el $w_1(x)$ va a ser rechazado por que “alguien mas nuevo leyó x1 antes que el”
- Como resuelve el control por timestamp el problema de la modificación perdida?

ÍNDICE

- Introducción
 - Lock binario, shared lock
 - Two phase locking
 - Control de concurrencia por timestamp
 - **Control de concurrencia por multiversion**
- Niveles Aislamiento en SQL

CONTROL DE CONCURRENCIA POR MULTIVERSIÓN

- La idea básica que esta detrás de este mecanismo de control de concurrencia, es la de darle a cada transacción la versión de los data ítems que estaban vigentes al momento de su inicio.
- Cuantas versiones es necesario guardar? En principio tantas como sean necesarias para atender a la transacción “ mas vieja” vigente. Solo se guardan las versiones correspondientes a transacciones activas o comiteadas (si una transacción aborto no es necesario guardar su versión)

CONTROL DE CONCURRENCIA POR MULTIVERSIÓN -CONT

- Sea $\{T_0, T_1, T_2 \dots T_N\}$ un conjunto de transacciones, para procesar las operaciones de este conjunto el scheduler necesita transformarlas en operaciones sobre versiones.
- Es necesario definir una función h tal que
- $h(w_i(x)) = w_i(x_i)$
- $h(r_i(x)) = r_i(x_j)$, donde T_j es la última transacción no abortada que escribió x .
- $h(a_j) = a_j$
- $h(c_j) = c_j$

MULTIVERSIÓN - CONT

- Una historia completa del tipo multiversión (MV) sobre H es un orden parcial con una relación de orden $<$ tal que
 - 1. $H = h(\bigcup_{i=1 \text{ to } n} T_i)$ para alguna función h ;
 - 2. para cada T_i y todas las operaciones p_i, q_i en T_i , si $p_i <_i q_i$, entonces $h(p_i) < h(q_i)$;
 - 3. si $h(r_j[x]) = r_j(x_i)$, entonces $w_i(x) < r_j(x)$;
 - 4. si $w_i[x] <_i r_i[x]$, entonces $h(r_i(x)) = r_i(x_i)$; y
 - 5. si $h(r_j[x]) = r_j(x_i)$, $i <> j$, y $c_j \in H$, entonces $c_i < c_j$.

MULTIVERSIÓN

- Cuando se rechaza una operación?
- Como seria el tratamiento de la falsa sumarización con este mecanismo?

ÍNDICE

- Introducción
 - Lock binario, shared lock
 - Two phase locking
 - Control de concurrencia por timestamp
 - Control de concurrencia por multiversion
- **Niveles Aislamiento en SQL**

NIVELES DE AISLAMIENTO EN SQL

Isolation level	Lost updates	Dirty reads	Non-repeatable reads	Phantoms
Read Uncommitted	don't occur	may occur	may occur	may occur
Read Committed	don't occur	don't occur	may occur	may occur
Repeatable Read	don't occur	don't occur	don't occur	may occur
Serializable	don't occur	don't occur	don't occur	don't occur

CONTROL DE CONCURRENCIA

- Esta presentación fue armada utilizando, además de material propio, material de
 - "Concurrency Control and Recovery in Database Systems" de Bernstein, Hadzilacos y Goodman



CONTROL DE CONCURRENCIA

MUCHAS GRACIAS!