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 (lost update)
  - Falsa Modificación (o dirty read)
  - Falsa Sumarizacion (non repeatable read)

MODIFICACIÓN PERDIDA

- Transacción 1
- r1(x)
- X = x + 2
- w1(x)

- Transacción 2
- r2(x)
- X= x +5
- w2(x)

FALSA MODIFICACIÓN

- Transacción 1
- r1(x)
- X = x + 2
- w1(x)
- a1

- Transacción 2
- r2(x)
- X= x +5
- w2(x)
- c2

FALSA SUMARIZACIÓN

- Transacción 1
- r1(x)
- r1(y)
- Y = y - 2
- X = x + 3
- w1(x)
- W1(y)

- Transacción 2
- r2(x)
- R2(y)
- Z = x + y
- W2 (z)

INTRODUCCIÓN

- 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
  - $l_i(x)$: Ti lockea el data ítem x
  - $u_i(x)$: Ti libera el lock sobre el data ítem x
- Cuando una transacción lee o escribe un ítem solicita un lock sobre el mismo
- 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

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 $pl_j(x)$ entran en conflicto si i<>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 Ti 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 Ti, 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 =rl1(x) , r1(x), u1(x), wl2(y), w2(y), u2(y),rl1(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 Ti a Tj cada vez hay un lock de Ti esperando a que se libere un lock de Tj. El arco se libera cuando se libera el ultimo lock de Tj que espera Ti
- 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) < qlj(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) < uqi(y)

TWO PHASE LOCKING – CORRECTITUD 2

- Usando esto vamos a demostrar que el grafo que produce el 2LPL es acíclico
- 1) Si Ti -> Tj esta en SG(H), entonces una de las operaciones de Ti en un data ítem, por ejemplo x, se ejecuto antes y entro en conflicto con una de las operaciones de Tj. Entonces Ti debe liberar el lock de x antes de que Tj pueda tomarlo.
- 2) Supongamos Ti -> Tj -> TK es un camino en SG(H), del primer paso se deduce que Ti libero algún lock antes de que Tj lo tomara y lo mismo vale entre Tj y Tk. Por lo tanto por transitividad Ti libero un lock antes de que Tk tomara uno. Por inducción esto puede extenderse arbitrariamente
- 3) Supongamos que SG(H) tiene un ciclo, digamos que T1->T2->……Tn->T1. Por el punto anterior esto implicaría que T1 libero un lock antes de que T1 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 pi(x) y qj(x) son dos operaciones en conflicto , entonces pi(x) < qj(x) si y solo si TS(Ti) < TS(Tj)
- Una forma de implementar el control por Timestamp es que el scheduler envia la operación al DM a menos que esta llegue "muy tarde".
- Una operación qi(x) "llega tarde" cuando el scheduler ya proceso oj(x) , con TS(Ti) < TS(Tj) y ambas operaciones están en conflicto. En este caso el scheduler aborta Ti 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?

TIMESTAMP - CONT

- Que operaciones controlan con que max? Por que?

TIMESTAMP - CONT

- 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
- Que pasa si luego de esto la operación cuyo max-w-scheduler use para un read aborta?

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X		
Y		
Z		

R1 (x)

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	1	
Y		
Z		

R1 (x)

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	1	
Y		
Z		

R1 (x)
W1(x)

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	1	1
Y		
Z		

R1 (x)
W1(x)

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	1	1
Y		
Z		

R1 (x)
W1(x)
R2(y)
W2(y)

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	1	1
Y	2	2
Z		

R1 (x)
W1(x)
R2(y)
W2(y)

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	1	1
Y	2	2
Z		

R1 (x)
W1(x)
R2(y)
W2(y)
R3(x)

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	3	1
Y	2	2
Z		

R1 (x)
W1(x)
R2(y)
W2(y)
R3(x)

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	3	1
Y	2	2
Z	3	3

R1 (x)
W1(x)
R2(y)
W2(y)
R3(x)
R3(z)
W3(z)

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	3	1
Y	2	2
Z	3	3

R1 (x)
W1(x)
R2(y)
W2(y)
R3(x)
R3(z)
W3(z)
R2(x)

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	3	1
Y	2	2
Z	3	3

R1 (x)
W1(x)
R2(y)
W2(y)
R3(x)
R3(z)
W3(z)
R2(x)
R2(z)

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	3	1
Y	2	2
Z	3	3

R1 (x)
W1(x)
R2(y)
W2(y)
R3(x)
R3(z)
W3(z)
R2(x)
R2(z)

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
  - R2(x) w2(x) w1(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 w1(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 w1(x) va a ser rechazado por que "alguien mas nuevo leyó x1 antes que el"

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, por que ?)

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp y que el valor inicial de cada data item es 0 y que cada transacción suma 1
- Tengo 3 data ítems

	Desde	Hasta	Valor
X	0		0
Y	0		0
Z	0		0

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp y que el valor inicial de cada data item es 0 y que cada transacción suma 1.
- Tengo 3 data ítems

R1 (x), lee 0
W1(x) , escribe 1

	Desde	Hasta	Valor
X	0	1	0
X	1		1
Y	0		0
Z	0		0

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp y que el valor inicial de cada data item es 0 y que cada transacción suma 1.
- Tengo 3 data ítems

R1 (x), lee 0
W1(x) , escribe 1
R2 (y), lee 0
W2(y) , escribe 1

	Desde	Hasta	Valor
X	0	1	0
X	1		1
Y	0	2	0
Y	2		1
Z	0		0

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp y que el valor inicial de cada data item es 0 y que cada transacción suma 1.
- Tengo 3 data ítems

R1 (x), lee 0
W1(x) , escribe 1
R2 (y), lee 0
W2(y) , escribe 1
r3(x), lee 1

	Desde	Hasta	Valor
X	0	1	0
X	1		1
Y	0	2	0
Y	2		1
Z	0		0

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp y que el valor inicial de cada data item es 0 y que cada transacción suma 1.
- Tengo 3 data ítems

R1 (x), lee 0
W1(x) , escribe 1
R2 (y), lee 0
W2(y) , escribe 1
r3(x), lee 1
R3 (z), lee 0
W3(z) , escribe 1

	Desde	Hasta	Valor
X	0	1	0
X	1		1
Y	0	2	0
Y	2		1
Z	0	3	0
Z	3		1

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp y que el valor inicial de cada data item es 0 y que cada transacción suma 1.
- Tengo 3 data ítems

R1 (x), lee 0
W1(x) , escribe 1
R2 (y), lee 0
W2(y) , escribe 1
r3(x), lee 1
R3 (z), lee 0
W3(z) , escribe 1
r2(x), lee 1

	Desde	Hasta	Valor
X	0	1	0
X	1		1
Y	0	2	0
Y	2		1
Z	0	3	0
Z	3		1

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp y que el valor inicial de cada data item es 0 y que cada transacción suma 1.
- Tengo 3 data ítems

R1 (x), lee 0
W1(x) , escribe 1
R2 (y), lee 0
W2(y) , escribe 1
r3(x), lee 1
R3 (z), lee 0
W3(z) , escribe 1
r2(x), lee 1
r2(z),

	Desde	Hasta	Valor
X	0	1	0
X	1		1
Y	0	2	0
Y	2		1
Z	0	3	0
Z	3		1

VEAMOS UN EJEMPLO

- R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp y que el valor inicial de cada data item es 0 y que cada transacción suma 1.
- Tengo 3 data ítems

R1 (x), lee 0
W1(x) , escribe 1
R2 (y), lee 0
W2(y) , escribe 1
r3(x), lee 1
R3 (z), lee 0
W3(z) , escribe 1
r2(x), lee 1
r2(z), **lee 0 y puede procesarlo**

	Desde	Hasta	Valor
X	0	1	0
X	1		1
Y	0	2	0
Y	2		1
Z	0	3	0
Z	3		1

CONTROL DE CONCURRENCIA POR MULTIVERSIÓN -CONT

- Sea {T0,T1, T2…TN} 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(wi(x)) = wi(xi)
- h(ri(x)) = ri(xj), donde Tj es la ultima transacción no abortada que escribió x.
- h(aj) = aj
- h(cj) = cj

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($U_{i=1 \text{ to } n}$Ti) para alguna función h;
  - 2. para cada Ti y todas las operaciones pi, qi en Ti, si pi <i qi, entonces h(pi) < h(qi);
  - 3. si h(rj[x]) = rj(xi), entonces wi(x) < rj(x);
  - 4. si wi[x] <i ri[x], entonces h(ri(x)) = ri[xi]; y
  - 5. si h(rj[x]) = rj(xi), i<> j, y cj E H, entonces ci < cj.

MULTIVERSIÓN

- Cuando se rechaza una operación?

OTRO PROBLEMA DE CONCURRENCIA

- Hay un problema mas sofisticado
  - Phantom Problem

Transacción 1

```
Select count(*)
From ventas
Where monto > 10.000;
```

Transacción 2

```
Insert into ventas (
…,monto) values (
….., 15.000)
```

PHANTOM PROBLEM

- Si el Schedule es T2 , T1 no hay problema
- Que pasa si es T1, T2?
- Como funcionan los mecanismos de control de concurrencia en este caso ?

PHANTOM PROBLEM

- Si el Schedule es T2 , T1 no hay problema
- Que pasa si es T1, T2?
- Como funcionan los mecanismos de control de concurrencia en este caso ?
- Los mecanismos de control de concurrencia no puede actuar sobre registros que no existen , de ahí el nombre de “phantom problema”

PHANTOM PROBLEM

- Para resolverlo se pueden usar diferentes alternativas
- Lockeo del predicado (no esta implementado en la practica, es NP-hard determinar si dos predicados son compatibles)
- Index locking
  - Lockea en forma compartida las “hojas” del índice que contiene los punteros a los datos que se leen
  - Permite lecturas, pero no borrados ni inserciones
- Si no se usa un índice para acceder se bloquean todas las hojas de un índice existente

REPASEMOS

	Share Lock	Timestamp	MultiVersion
Falsa modificación	1	1	1
Modificación Perdida	2	2	2
Falsa Sumarización	3	3	3

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!